

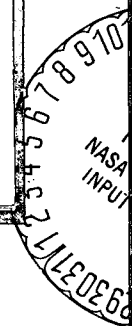
ELECTRICAL

E
N
G
-
N
E
E
R
-
N
G

(NASA-CR-124166) SURVEY OF DIGITAL
FILTERING Final Technical Report, 17
Jun. 1965 - 15 Oct. 1972 (Auburn Univ.)
411 p HC \$22.75
CSCI 09C

Unclas
G3/10 17350

N73-20256



Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
US Department of Commerce
Springfield, VA. 22151

ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY

AUBURN, ALABAMA

AUBURN UNIVERSITY

AUBURN



ALABAMA

36830

SCHOOL OF ENGINEERING

Electrical Engineering
207 Dunstan Hall

Telephone 826-4330
Area Code 205

March 20, 1973

National Aeronautics and Space Administration
George C. Marshall Space Flight Center
Huntsville, Alabama 35812

RE: Contract NAS8-26580
Monthly Progress Report
February 5, 1973 to
March 5, 1973

Dear Sir:

Progress in the assigned area of work is as follows:

Feasibility Study of Reaction Control Systems
(Huntsville Contact: C. Rupp)
M. Polites

Task 1.

Additional effort is being devoted to formalizing a method of estimating the error-time response for the class of MRAS systems studied previously. In addition, a general set of conditions governing a class of coupled MRAS systems is being outlined.

Task 2. (a). Using a torque axis transformation design procedure a new 16 engine model has been developed and is presently under study.

(b). Study of the optimal CMG control law is continuing.

Sincerely yours,

Joseph S. Boland, III
Joseph S. Boland, III
Project Leader

JSB/vht

I

FINAL TECHNICAL REPORT
SURVEY OF DIGITAL FILTERING

PREPARED BY

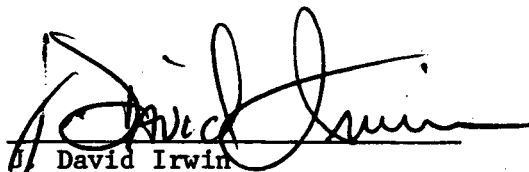
DIGITAL SYSTEMS LABORATORY
ELECTRICAL ENGINEERING

H. TROY NAGLE, JR
PROJECT LEADER


OCTOBER, 1972

CONTRACT NAS8-20163
GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HUNTSVILLE, ALABAMA

APPROVED BY:


David Irwin
Associate Professor and Head
Electrical Engineering

SUBMITTED BY:


H. Troy Nagle, Jr.
Associate Professor
Electrical Engineering

PRECEDING PAGE BLANK NOT FILMED

FOREWORD

This is a technical summary reporting the progress of a study conducted by the Electrical Engineering Department of Auburn University during the period 17 June 1965 through 15 October 1972. This study report completes Contract No. NAS8-20163, granted to Engineering Experiment Station, Auburn, Alabama, by George C. Marshall Space Flight Center, National Aeronautics and Space Administration, Huntsville, Alabama.

ACKNOWLEDGEMENT

The author wishes to recognize the following list of Faculty and Graduate Students who participated in this study during its various stages

Faculty:

Chester C. Carroll, Project Leader, 1965-1970
H. Troy Nagle, Jr., Project Leader, 1970-1972

Graduate Students:

J. R. Heath	Quitman Liner
Ronald White	Roger Cole
J. W. Jones	J. L. Raley
W. L. Oliver	John A. Childs
G. E. Jordan	David Kimsey
H. H. Hull	LeRoy Bearnson
H. Troy Nagle, Jr.	R. H. Robison

SURVEY OF DIGITAL FILTERING

H. Troy Nagle, Jr.

ABSTRACT

A three part survey is made of the state-of-the-art in digital filtering. Part one presents background material including sampled-data transformations and the discrete Fourier transform. Part two, digital filter theory, gives an in-depth coverage of filter categories, transfer function synthesis, quantization and other non-linear errors, filter structures and computer aided design. Part three presents hardware mechanization techniques. Implementation by general-purpose, mini-, and special-purpose computer are presented.

SURVEY OF
DIGITAL FILTERING

<u>Part</u>	<u>Page</u>
1. Background	1-1
2. Digital Filter Theory	2-1
3. Mechanization of Digital Filters	3-1

PART ONE

BACKGROUND

PART ONE: BACKGROUND

TABLE OF CONTENTS

I.	Introduction to Digital Filtering	1-1
A.	The Computer Model.	1-1
B.	The z-Domain Model.	1-4
C.	Scope of Digital Filtering.	1-4
II.	Standard z-Transform.	1-7
A.	Impulse Sampling.	1-7
B.	Hold Devices.	1-11
C.	Discrete Transfer Functions	1-14
D.	Difference Equations.	1-16
E.	Mapping Function.	1-19
F.	Frequency Response.	1-22
III.	Sampled Data Transformations.	1-23
A.	Approximation Techniques.	1-23
1.	Backward Difference	1-23
2.	Forward Difference.	1-28
3.	Rectangular Rule.	1-30
a.	Left Side	1-30
b.	Right Side.	1-32
4.	Trapezoidal Rule.	1-33
5.	Simpson's Rule.	1-34
6.	Impulse Invariance.	1-34
7.	Impulse Invariant Integrator.	1-36
B.	Mapping Functions Summary	1-38
1.	Standard z-Transform.	1-38
2.	Backward Difference	1-38
3.	Forward Difference.	1-40
4.	Bilinear z-Transform.	1-40
5.	Matched z-Transform	1-45
C.	Other Transforms.	1-46
1.	Simpson's Rule.	1-46
2.	(w, v)-Transform.	1-46
IV.	Discrete State Variables.	1-48

V.	Convolution.	1-50
	A. Continuous Linear Systems.	1-50
	B. Discrete Linear Systems.	1-52
VI.	Discrete Fourier Transform	1-54
	A. Continuous Fourier Transform	1-54
	B. Discrete Fourier Transform	1-54
	1. Sampling Process	1-54
	2. DFT Derivation	1-58
	3. IDFT Derivation.	1-61
	4. DFT Pairs.	1-62
	C. Fast Fourier Transform	1-64
	1. Calculation Time	1-64
	2. FFT Derivation	1-67
	3. IFFT Derivation.	1-68
VII.	Random Processes	1-73
		1
	A. Continuous Processes	1-73
	B. Discrete Processes	1-73
	REFERENCES	1-76

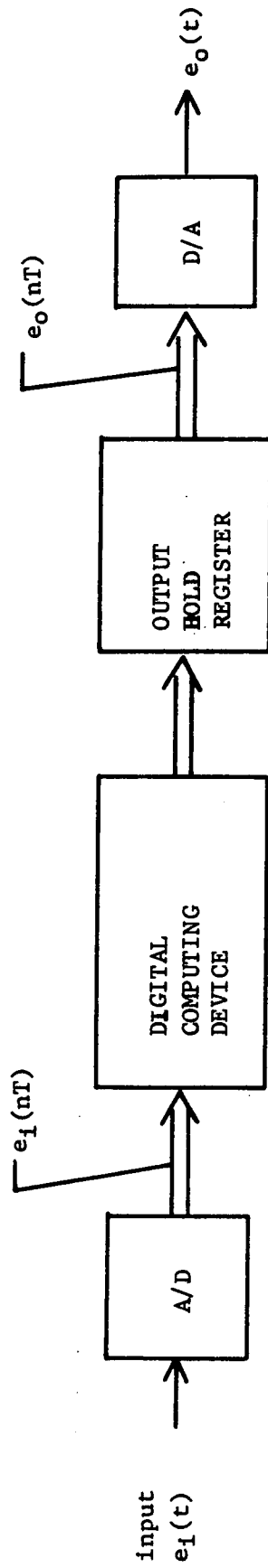
I. INTRODUCTION TO DIGITAL FILTERING

Digital Filtering may be described as the process by which input discrete-time sequences of numbers with discrete amplitudes are transformed into output discrete-time sequences of numbers with discrete amplitudes. The transformation process (the digital filter) may be described as a set of difference equations which may be programmed on a general-purpose computer, or realized with specially designed devices.

The Computer Model

An example digital filter is shown in Fig. 1. The input signal $e_i(t)$ is in analog form and is sampled every T seconds by an Analog-to-Digital Converter (A/D). The input samples $e_i(nT)$, n an integer, are in binary 2's complement form and are supplied to the computing device, which may be a general-purpose or special-purpose computer. The computing device is programmed to calculate the filter output samples $e_o(nT)$ which are fed to an output hold register. This register may actually be considered to be part of the computing device. The output samples $e_o(nT)$ are held in the register until a new output sample is calculated and supplied to the output hold register. The D/A converter produces an analog output signal $e_o(t)$ whose characteristic form is shown in Fig. 2.

The digital filter of Fig. 1 operates as follows: A pulse from the digital filter control unit at $t=nT$ instructs the A/D to calculate $e_i(nT)$. However this sampled value of $e_i(t)$ is not available to the computing device until time $nT + T_a$, where T_a is the total A/D



NOTE: SINGLE LINES REPRESENT ANALOG SIGNALS
DOUBLE LINES REPRESENT DIGITAL SIGNALS

Fig. 1. A digital filter

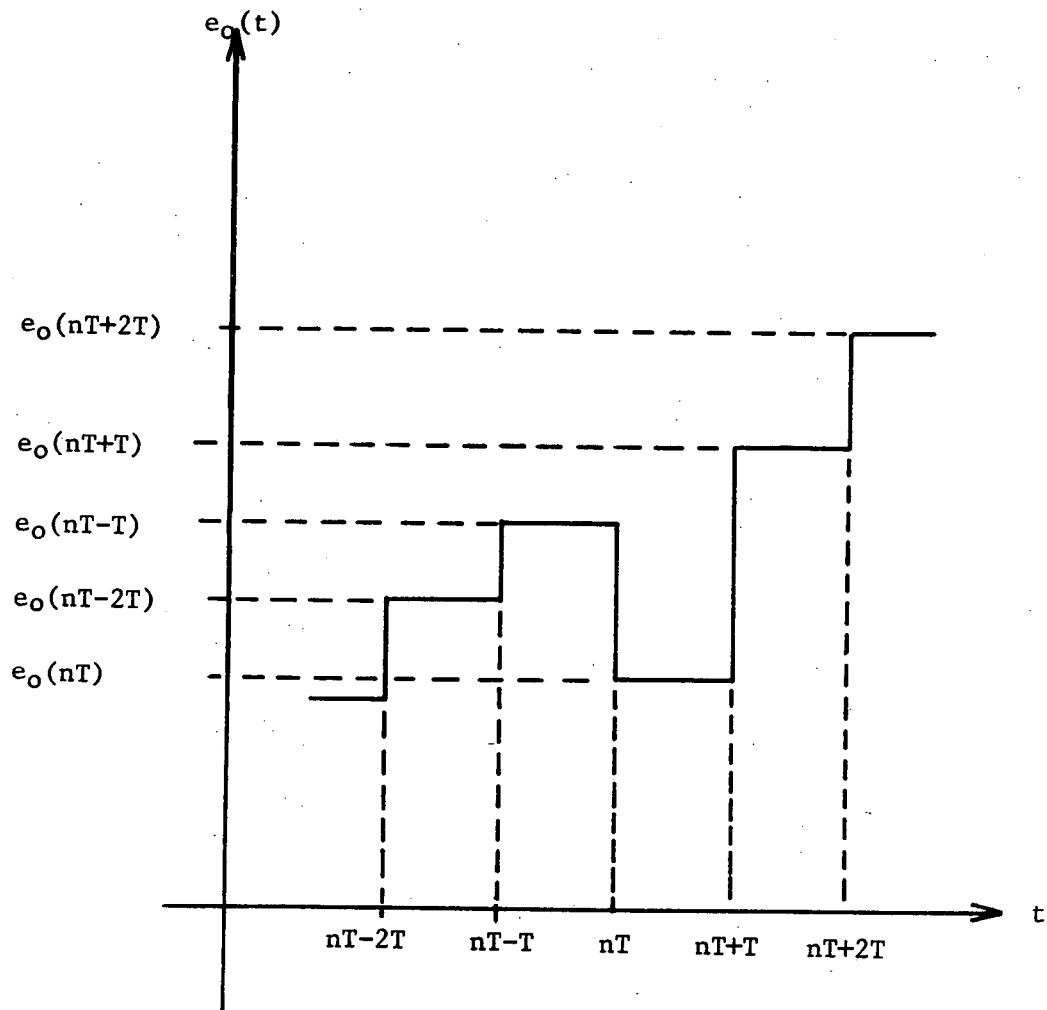


Fig. 2. The analog output $e_o(t)$.

conversion time. Once the input sample has arrived at the computing device, the output sample is calculated and sent to the output hold device at $t=nT+T_a+T_c$, where T_c is the computing time. Thus the output $e_o(nT)$ is actually $e_o(nT+T_a+T_c)$. With modern technology, T_a+T_c can be designed to take less than $1\mu s$. Hence for sampling rates of up to 200KHz ($T=5\mu s$), T is much greater than T_a+T_c and hence,

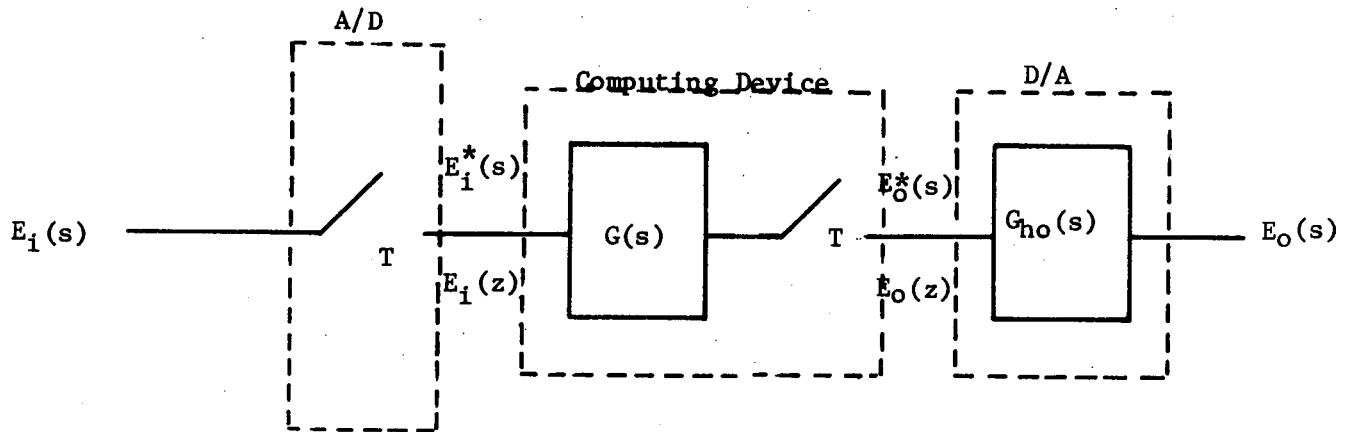
$$e_o(nT+T_a+T_c) \approx e_o(nT).$$

The z-Domain Model

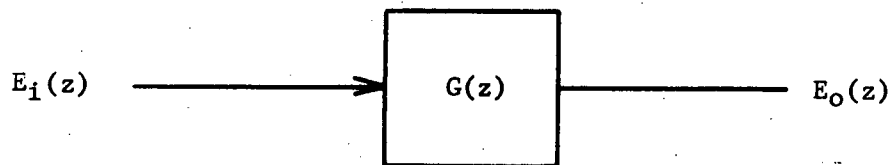
The digital filter of Fig. 1 has been examined and described from a hardware or functional point of view. A mathematical model for this filter is shown in Fig. 3 which employs the well known z-transform. Fig. 3a demonstrates a discrete time model for the digital filter where the switches labeled T represent impulse samplers and the block labeled $G_{ho}(s)$ represents a "zero-order hold" device. Fig. 3b illustrates the transfer function representation of the computing device itself. Fig. 3 differs from Fig. 1 in that the computing device of Fig. 1 uses the amplitude of the input (and output) samples to calculate new output samples $e_o(nt)$; Fig. 3 uses impulse functions weighted by the amplitude of the input (and output) samples to calculate new output impulses $e_o^*(t)$. The impulse samples, zero-hold, and z-transform will be discussed in more detail later.

Scope of Digital Filtering

The digital filter models presented above were for conventional one-dimensional processing of a single input variable. Although this concept of digital filtering is most widely accepted, many other researchers



(a) Model of entire filter



(b) Model of computing device

Fig. 3. Mathematical model of
the digital filter.

have applied the label to more general schemes. Digital filtering in the last few years has come to also mean optimal state estimation, discrete Fourier transformation, high speed convolution, non-linear discrete filtering, two-dimension image processing, random and multirate sampling, block recursion, least-mean squares filtering, quantization optimization, computer programming, and hardware implementation. All of these topics, and others, will be introduced in what follows. Emphasis will be, however, on the standard case of linear, one-dimensional digital filtering. A prerequisite to understanding the theory of digital filtering is a mathematical background in sampled-data transforms, Fourier transforms, convolution, discrete state variables and stochastic processes. Hence, these topics are now reviewed briefly.

II. STANDARD Z-TRANSFORM

The most common sampled-data transformation is called the standard z-transform. It is used to describe both the sampling process for the digital filter input signal and the discrete transfer function of the filter itself.

Impulse Sampling

The z-transform is used to represent mathematically a discrete-time system. The discrete-time intervals are produced by periodic impulse samplers. Consider Fig. 4. The Laplace transfer function $G(s)$ is an analog filter; its input is a Dirac delta function. The filter output $g(t)$ is periodically impulse sampled every T seconds. The sampled output may be expressed as

$$\begin{aligned} g^*(t) &= g(t) \sum_{k=0}^{\infty} \delta(t-kT) \\ &= \sum_{k=0}^{\infty} g(kT) \delta(t-kT) \end{aligned} \tag{1a}$$

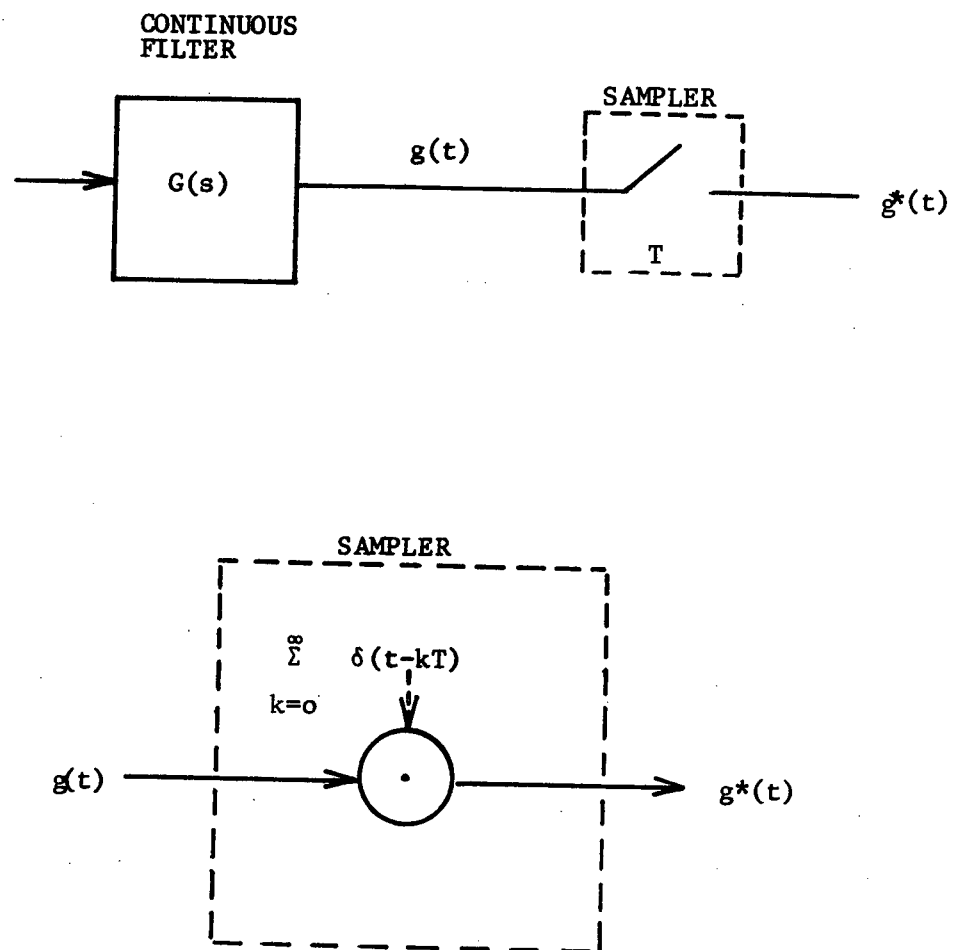


Fig. 4. A continuous filter and sampler.

The Laplace transform of $g^*(t)$ is

$$\mathcal{L}[g^*(t)] = G^*(s) = \sum_{k=0}^{\infty} g(kT)e^{-kTs}$$

If we define $z = e^{Ts}$, then

$$G^*(s) \bigg|_{s = \frac{\ln z}{T}} = \sum_{k=0}^{\infty} g(kT)z^{-k} \quad (1b)$$

Equation (1b) is the standard z -transform of $g(t)$, or

$$G(z) \triangleq Z[g(t)] \triangleq G^*(s) \bigg|_{s = \frac{\ln z}{T}} = \sum_{k=0}^{\infty} g(kT)z^{-k} \quad (2)$$

Suppose that the analog transfer function $G(s)$ is of the form

$$G(s) = K \frac{\prod_{i=1}^m (s+a_i)}{\prod_{j=1}^n (s+b_j)} \quad (3a)$$

Where

$-a_i$ = complex zeroes of $G(s)$

$-b_j$ = complex poles of $G(s)$

$s = \sigma + j\omega$ = Laplace variable

K = Constant

$n > m$

If there are no repeated poles in $G(s)$, then

$$G(s) = \sum_{k=1}^n \frac{R_k}{s+b_k} \quad (3b)$$

Where R_k is the residue at pole $-b_k$.

Since,

$$Z\left[\frac{a}{s+u}\right] = Z\left[ae^{-ut}\right] = \frac{a}{1-e^{-uT}z^{-1}}$$

The standard z-transform of (3b) results in

$$G(z) = \sum_{k=1}^n \frac{R_k}{1-e^{-b_k T}z^{-1}} \quad (4)$$

A third representation of interest is found by noting that multiplication of $g(t)$ and $\sum_{k=0}^{\infty} \delta(t-kT)$ in the time domain corresponds to convolution of $G(s)$ and $\sum_{k=0}^{\infty} e^{-kTs} = \frac{1}{1-e^{-Ts}}$ in the frequency domain. After convolution the result is

$$G^*(s) = 1/2 \cdot g(0+) + 1/T \sum_{k=-\infty}^{\infty} G(s+jk\omega_g) \quad (5)$$

where $\omega_s = 2\pi/T$. Hence it is apparent that $G^*(s)$ is periodic in ω , the sampling frequency. It is required that $G(j\omega) = 0$ for $|\omega| > \omega_s/2$ as shown in Fig. 5 so that the frequency content of $G(j\omega)$ will be preserved in the primary strip of $G^*(j\omega)$. If this relationship is preserved the envelope of $G(j\omega)$ can be recovered from $G^*(j\omega)$. This phenomenon is known as frequency aliasing.

The standard z-transform discussed above is best known of the sampled-data transforms. The theorems and tables of z-transforms can be found in any standard sampled-data text [1].

Hold Devices

In the previous section we have seen the sampling process used to determine values of an input signal at discrete time intervals. The inverse process of data reconstruction from these sampled signals is accomplished by hold devices. Consider the problem of reconstructing the signal $g(t)$ given samples spaced T seconds apart. If we expand $g(t)$ in a Taylor's series

$$g(t) = g(nT) + g'(nT)(t-nT) + \frac{g''(nT)}{2!} (t-nT)^2 + \dots \quad (6)$$

for $nT \leq t < (n+1)T$,

where $g'(t) = \frac{dg(t)}{dt}$.

The derivatives may be approximated by

$$g'(nT) = \frac{1}{T} (g(nT) - g(nT-T))$$

$$g''(nT) = \frac{1}{T} (g'(nT) - g'(nT-T)) , \quad \text{etc.} \quad (7)$$

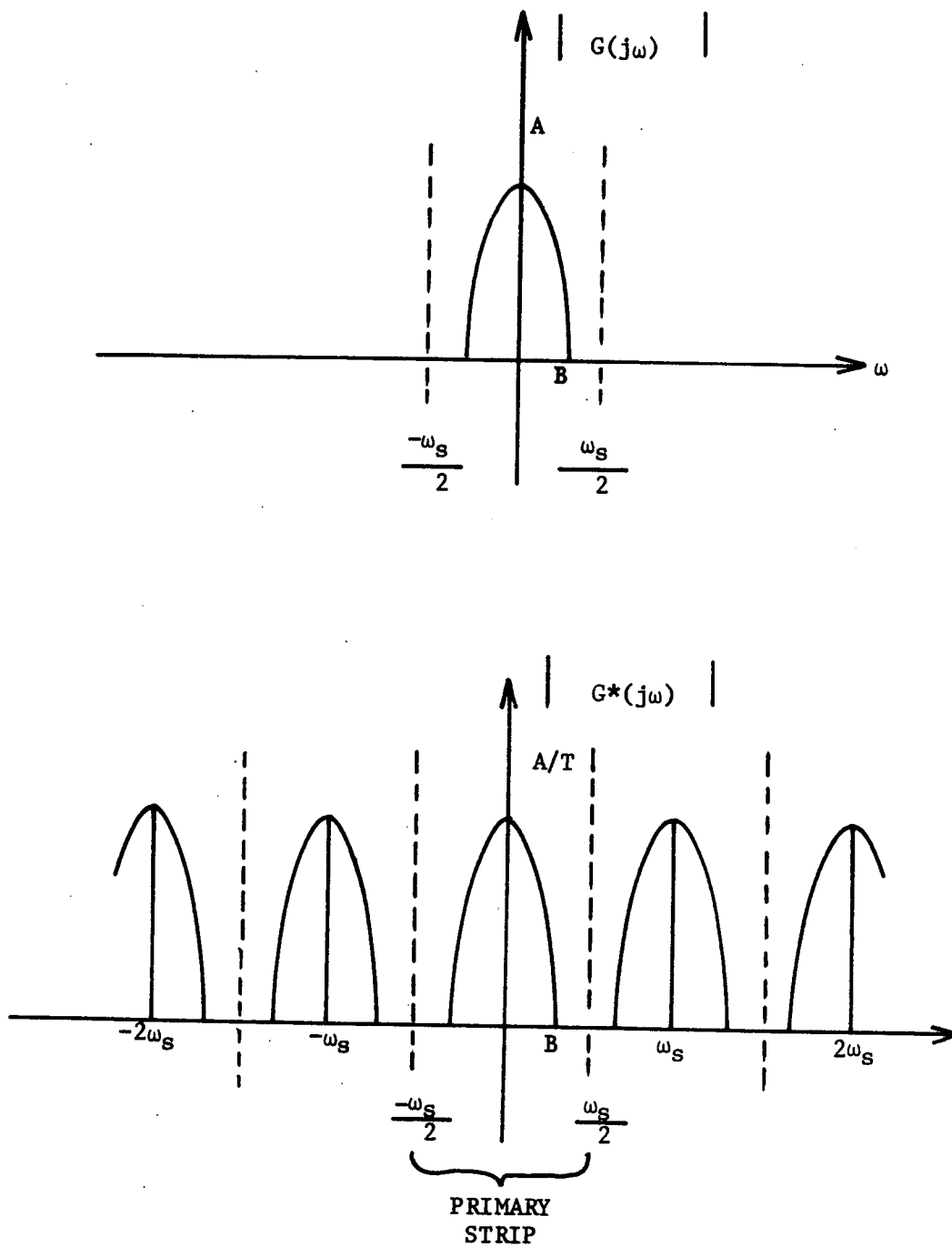


Fig. 5. Frequency domain characteristics of $G^*(s)$.

If equation (6) is truncated to just one term, this reconstruction is called a zero-order hold; if the first derivative is included, a first-order hold; etc.

Zero-Order Hold [1]

The zero-order hold device in the mathematical model of Fig. 3 accepts an impulse modulated input $e_o^*(t)$ and produces an output $e_o(t)$ as shown in Fig. 2. The input $e_o^*(t)$ may be expressed as

$$e_o^*(t) = \sum_{k=0}^{\infty} e_o(kT) \delta(t-kT).$$

Its Laplace transform is

$$E_o^*(s) = E_o(z) = \sum_{k=0}^{\infty} e_o(kT) z^{-k}.$$

The output (see Fig. 2) may be written

$$e_o(t) = \sum_{k=0}^{\infty} e_o(kT) [u(t-kT) - u(t-kT-T)]. \quad (8)$$

Its Laplace transform is

$$\begin{aligned} E_o(s) &= \sum_{k=0}^{\infty} e_o(kT) \left[\frac{e^{-kTs}}{s} - \frac{e^{-kTs} e^{-Ts}}{s} \right] \\ &= \sum_{k=0}^{\infty} e_o(kT) e^{-kTs} \left(\frac{1 - e^{-Ts}}{s} \right) \\ &= E_o(z) \left(\frac{1 - e^{-Ts}}{s} \right). \end{aligned}$$

The transfer function of the zero-order hold device is

$$G_{ho}(s) = \frac{E_o(s)}{E_o^*(s)} = \frac{1-e^{-Ts}}{s}. \quad (9)$$

The frequency domain characteristics of $G_{ho}(s)$ are shown in Fig. 6.

Suppose that the sampling interval T is chosen very small. Then,

$$\begin{aligned} e^{-Ts} &= 1-Ts + O(T^2) \\ &\approx 1-Ts \end{aligned}$$

Then,

$$G_{ho}(s) \approx \frac{1-(1-Ts)}{s} = T. \quad (10)$$

This result is verified by noting in Fig. 6 that, for $\omega \ll \omega_s/2$ (or T small), the magnitude function approaches T ; also, the zero-order hold introduces phase lag which is linear with frequency into the system.

Discrete Transfer Functions [1]

In the mathematical model of Fig. 3, the transfer function of the computing device is shown as

$$\frac{E_o(z)}{E_i(z)} = \frac{E_o^*(s)}{E_i^*(s)} = G(z).$$

From Fig. 3

$$E_o(s) = E_i^*(s)G(s).$$

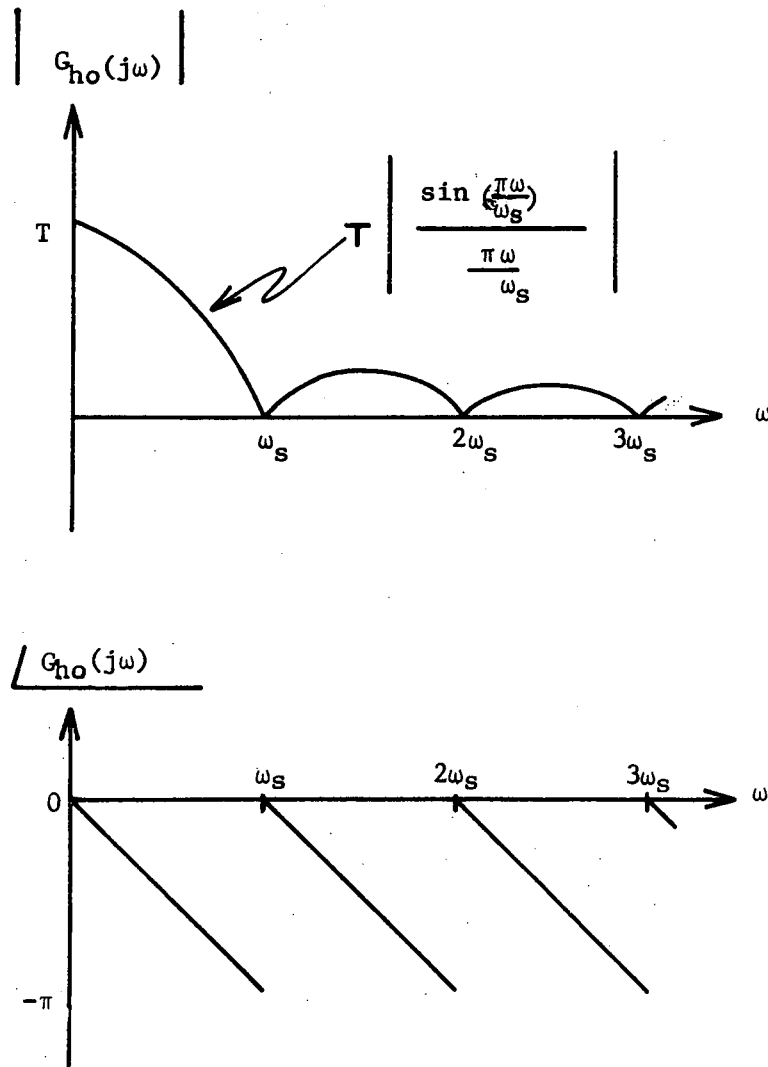


Fig. 6. Gain and phase characteristics of a zero-order hold.

Hence,

$$e_o^*(t) = \left[\sum_{k=0}^{\infty} e_1(kT) g(t-kT) \right] \sum_{j=0}^{\infty} \delta(t-jT).$$

In this expression $g(t-kT)$ may be replaced with $g(jT-kT)$ due to the Dirac delta function. Next, let $\ell=j-k$ and replace the summation index j with ℓ as follows:

$$e_o^*(t) = \sum_{\ell=-k}^{\infty} \sum_{k=0}^{\infty} e_1(kT) g(\ell T) \delta(t-\ell T-kT).$$

Since $g(y) = 0$ for $y < 0$, the $-k$ may be replaced by zero. The Laplace transform of $e_o^*(t)$ is

$$\begin{aligned} E_o^*(s) &= \left[\sum_{k=0}^{\infty} e_1(kT) e^{-kTs} \right] \left[\sum_{\ell=0}^{\infty} g(\ell T) e^{-\ell Ts} \right] \\ &= E_1^*(s) G^*(s). \end{aligned}$$

Hence, the discrete-time transfer function of the computing device in Fig. 3 is indeed

$$\frac{E_o^*(s)}{E_1^*(s)} = G^*(s) = G(z). \quad (11)$$

Difference Equations [4]

The discrete transfer function of equation (11) is, in general, the ratio of two polynomials in z^{-1} ,

$$G(z) = \frac{a_0 + a_1 z^{-1} + \dots + a_n z^{-n}}{1 + b_1 z^{-1} + \dots + b_n z^{-n}} = \frac{E_o(z)}{E_1(z)} \quad (12)$$

where the coefficients a_i and b_j are real numbers (can be zero). An equivalent expression for (12) is the equation

$$E_0(z) = a_0 E_1(z) + a_1 z^{-1} E_1(z) + \dots + a_n z^{-n} E_1(z) \\ - b_1 z^{-1} E_0(z) - \dots - b_n z^{-n} E_0(z).$$

The infinite series for the z-transforms $E_0(z)$ and $E_1(z)$ is now substituted into the above equation and the coefficients of like powers of z^{-1} are equated, yielding

$$e_0(kT) = a_0 e_1(kT) + a_1 e_1(kT-T) + \dots + a_n e_1(kT-nT) - b_1 e_0(kT-T) - \dots \\ - b_n e_0(kT-nT). \quad (13)$$

Note that $z^{-1} = e^{-Ts}$ which represent a time delay of T second. Equation (13) may be programmed in the computing device of Fig. 1; the variable $e_1(kT)$ is furnished by the A/D converter; delayed values of the input $e_1(kT-nT)$ and delayed values of the output variable $e_0(kT-nT)$ are stored in the computing device.

Equation (13) defines a programming scheme known as the direct form. A block diagram of this form appears in Fig. 7.

Another programming scheme known as the canonical form is determined below. The transfer function is expressed as

$$G(z) = \frac{E_0(z)}{M(z)} = \frac{M(z)}{E_1(z)}$$

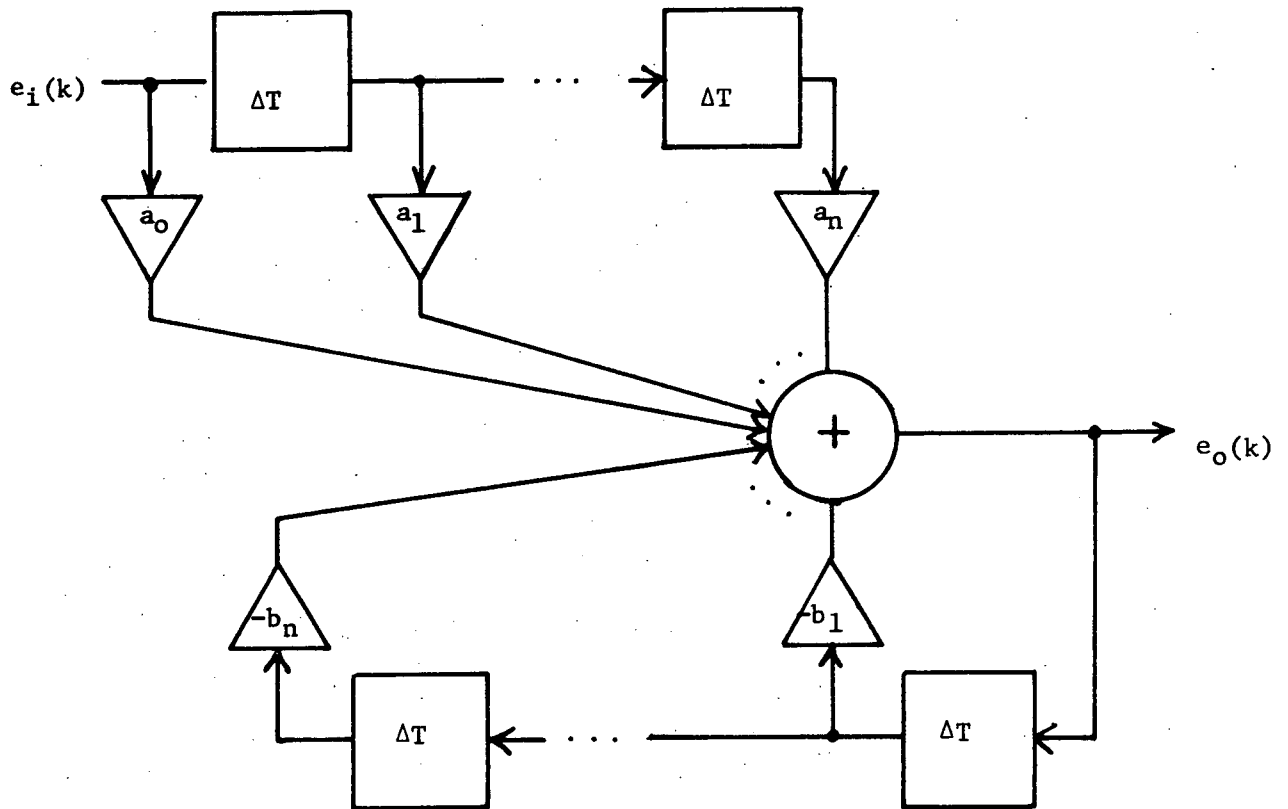


Fig. 7. Generalized block diagram of the direct programming form for a digital filter.

where

$$\frac{E_o(z)}{M(z)} = a_o + a_1 z^{-1} + \dots + a_n z^{-n}$$

$$\frac{M(z)}{E_i(z)} = \frac{1}{1 + b_1 z^{-1} + \dots + b_n z^{-n}} \quad (14)$$

The time-domain equivalent expressions for (14) are

$$m(kT) = e_i(kT) - b_1 m(kT-T) - \dots - b_n m(kT-nT) \quad (15)$$

$$e_o(kT) = a_o m(kT) + a_1 m(kT-T) + \dots + a_n m(kT-nT) \quad (16)$$

Equations (15) and (16) are the difference equations to be used in the canonical programming form. A generalized block diagram of this form appears in Fig. 8.

Mapping Function

The standard z-transformation of an analog function in the s-plane may be considered to be a mapping from the s-plane to the z-plane under the rule

$$z = e^{Ts}. \quad (17)$$

See Fig. 9. The mapping illustrates that the region of stability in the s-plane (the left half-plane) corresponds to the interior of the unit circle in the z-plane. In fact, the primary strip in the s-plane maps onto the unit circle. All other strips also map to the unit circle

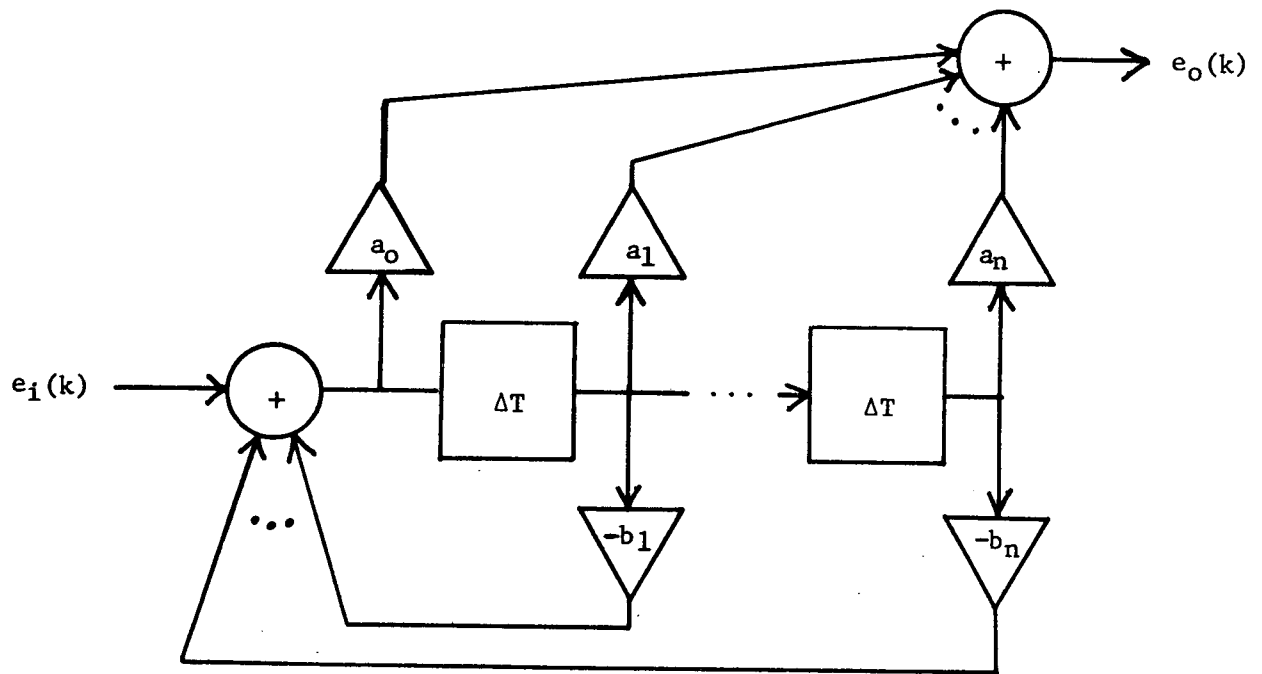


Fig. 8. Generalized block diagram of the canonical programming form for a digital filter.

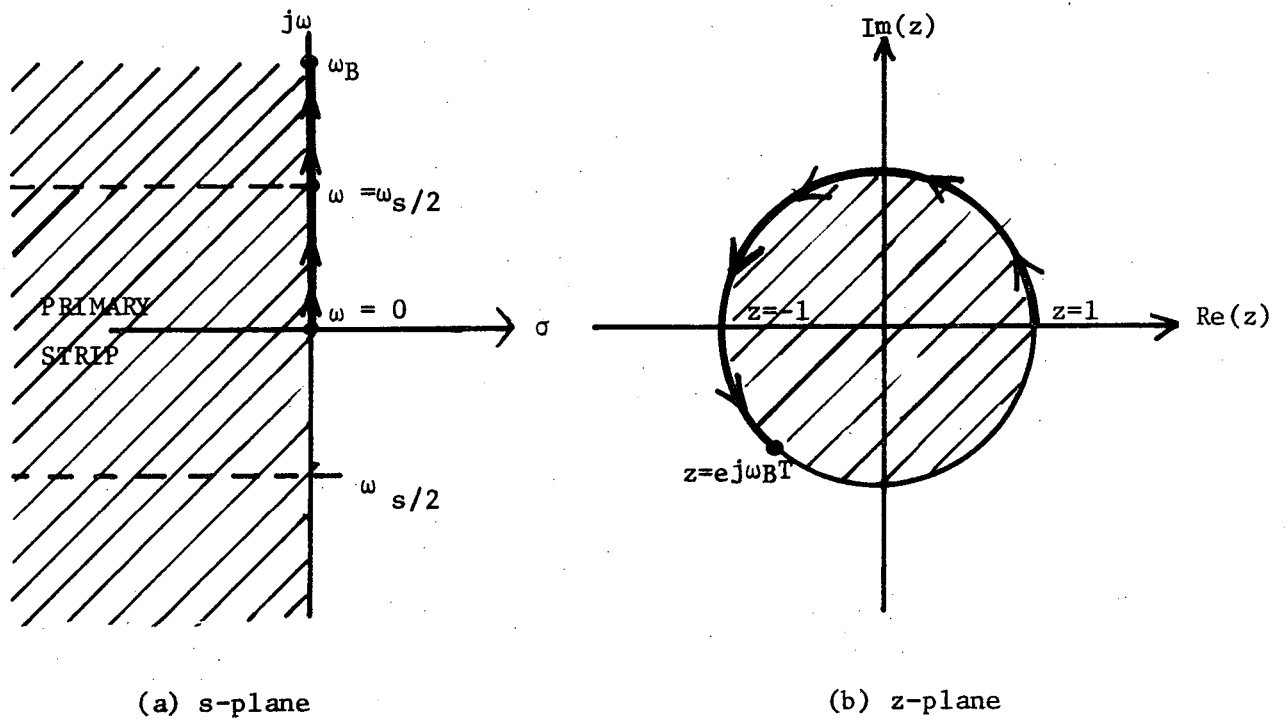


Fig. 9. s-plane to z-plane mapping.

further illustrating the frequency aliasing problem of Fig. 5. Thus, transfer functions which are stable in the s-plane will also be stable after taking their standard z-transformation.

Frequency Response

The frequency response in the s-plane is evaluated by

$$|G(s)|_{s=j\omega}$$

$$\angle G(s)_{s=j\omega} \quad 0 \leq \omega < \omega_B .$$

This corresponds to evaluating $G(s)$ along the contour in the s-plane of Fig. 9a. Some upper cutoff frequency ω_B is shown for illustration. In the z-domain the contour follows the unit circle so that

$$|D(z)|_{z=e^{j\omega T}}$$

$$\angle D(z)_{z=e^{j\omega T}} \quad 0 \leq \omega < \omega_B ,$$

is used to calculate the frequency response of a discrete transfer function.

From equation (5) we see that $D(z)$ is periodic in ω_s so that in practice

$\omega_B = \omega_s/2$ and the contour traverses the top half of the unit circle. Hence

$$\begin{aligned} \text{db} &= 20 \log |D(e^{j\omega T})| \\ \phi &= \angle D(e^{j\omega T}) \end{aligned} \quad 0 \leq \omega \leq \omega_s/2 \quad (18)$$

will be used to calculate the frequency response of a digital filter.

III. SAMPLED DATA TRANSFORMATIONS

Sampled-data transformations are the techniques one uses to obtain numerical solutions to integral and differential equations. Any linear system's transfer function may be written as

$$G(s) = \frac{Y(s)}{X(s)}$$

$Y(s)$ = Laplace transform of the output

$X(s)$ = Laplace transform of the input.

Alternately the relationship between input and output may be described as a differential or integral equation. Numerical methods may be employed to solve these equations; these methods approximate the integral and differential equations by difference equations. As we have seen previously the difference equations may be represented by a discrete transfer function. The complete process is illustrated in Figure 10.

Numerical Approximations

Several numerical approximation techniques will now be presented, some for differentiation and some for integration.

Backward Difference

The backward difference is a simple technique which replaces the derivative of a function by

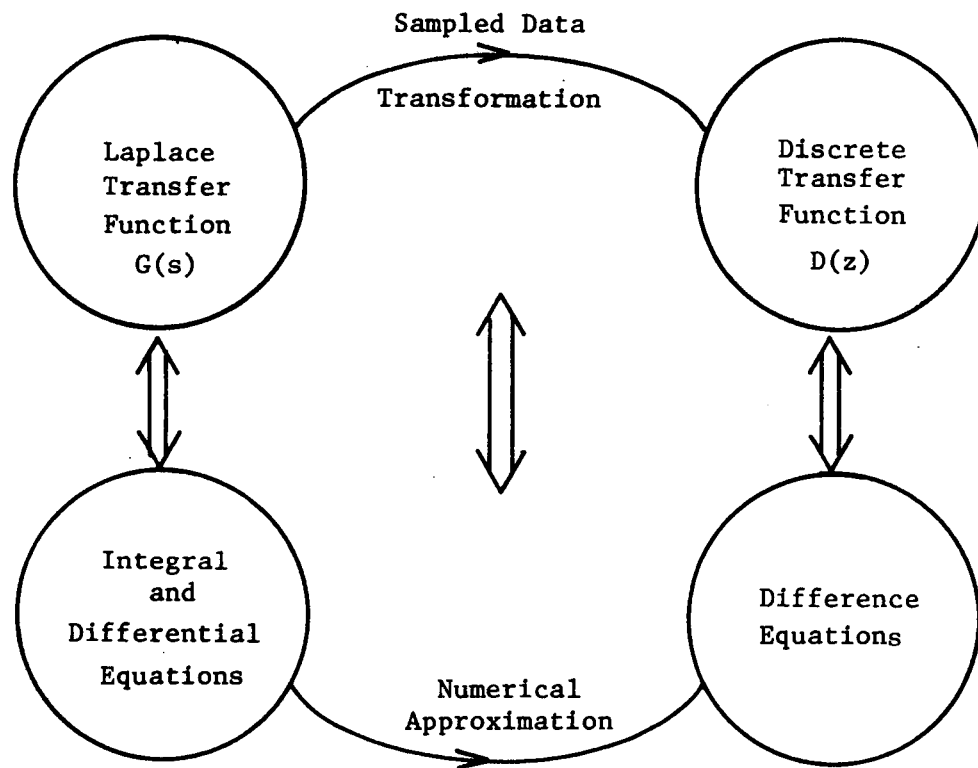


Figure 10. Relation between numerical approximations and sampled data transformations.

$$\frac{d}{dt} y(t) \doteq \frac{y(t) - y(t - T)}{T} .$$

See Figure 11.

In the Laplace domain

$$sY(s) \doteq \frac{Y(s) - e^{-sT}Y(s)}{T}$$

$$s \doteq \frac{1 - e^{-sT}}{T}$$

$$s \doteq \frac{1 - z^{-1}}{T} .$$

Hence,

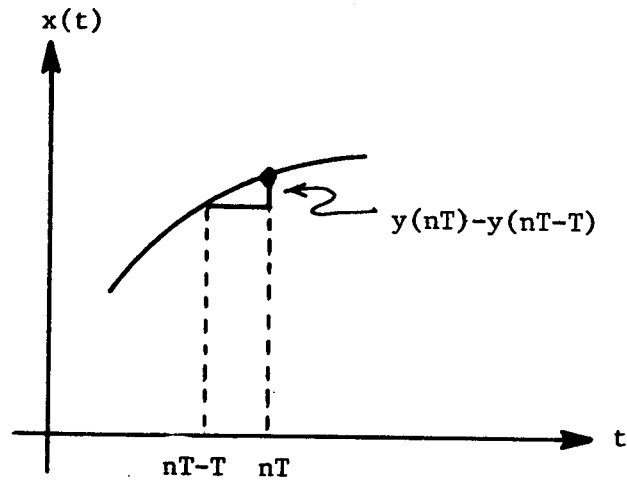
$$D(z) = G(s) \Big|_{s = \frac{1 - z^{-1}}{T}} . \quad (19)$$

Example. Find a discrete approximation for

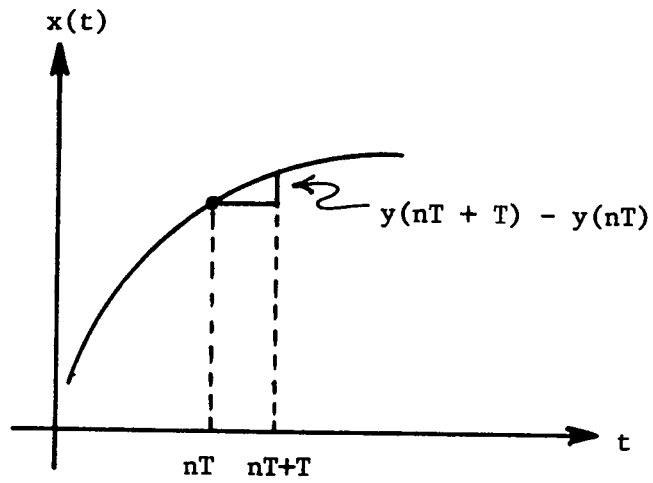
$$G(s) = \frac{s}{s + a}$$

$$Y(s) = G(s) X(s)$$

$$sY(s) + aY(s) = sX(s)$$



(a) Backward Difference



(b) Forward Difference

Figure 11. Difference Approximations.

or

$$\frac{d}{dt} y(t) + ay(t) = \frac{d}{dt} x(t).$$

Now let

$$\frac{d}{dt} y(t) = \frac{y(t) - y(t - T)}{T}.$$

$$\frac{d}{dt} x(t) = \frac{x(t) - x(t - T)}{T}.$$

Therefore

$$\frac{y(t) - y(t - T)}{T} + ay(t) = \frac{x(t) - x(t - T)}{T}.$$

Evaluating at $t = nT$ yields

$$y(nT) = \frac{1}{1 + Ta} (x(nT) - x(nT - T) + y(nT - T)).$$

Employing equations (12) and (13),

$$D(z) = \frac{1}{1 + Ta} \frac{1 - z^{-1}}{1 - \frac{1}{1 + Ta} z^{-1}}.$$

An alternate solution employs equation (19) as follows

$$D(z) = \frac{s}{s + a} \quad \left| \quad s = \frac{1 - z^{-1}}{T} \right.$$

$$= \frac{\frac{1 - z^{-1}}{T}}{a + \frac{1 - z^{-1}}{T}}$$

$$= \frac{1 - z^{-1}}{aT + 1 - z^{-1}}$$

$$D(z) = \frac{1}{1 + aT} \cdot \frac{1 - z^{-1}}{1 - \frac{1}{1 + aT} z^{-1}}$$

Forward Difference

A similar numerical technique approximates

$$\frac{d}{dt} y(t) \doteq \frac{y(t + T) - y(t)}{T} \quad .$$

See Figure 11.

This represents the equivalent Laplace domain approximation

$$sY(s) \doteq \frac{e^{sT}Y(s) - Y(s)}{T}$$

or

$$s \doteq \frac{e^{sT} - 1}{T}$$

$$s \doteq \frac{z - 1}{T}$$

Hence,

$$D(z) = G(s) \bigg|_{s = \frac{z - 1}{T}} \quad (20)$$

Example. Find a discrete version of $G(s)$ using the forward difference.

$$G(s) = \frac{s}{s + a}$$

$$D(z) = \frac{s}{s + a} \bigg|_{s = \frac{z - 1}{T}}$$

$$D(z) = \frac{\frac{z-1}{T}}{\frac{z-1}{T} + a}$$

$$D(z) = \frac{1 - z^{-1}}{1 + (aT - 1) z^{-1}}$$

Rectangular Rule

Suppose now we try some numerical approximations to integrals and compare results.

Left Side Rule. Let us determine the numerical approximation for

$$y(t) = \int_0^t x(t) dt .$$

Assume that the upper limit of the integral is $t = nT$. Hence

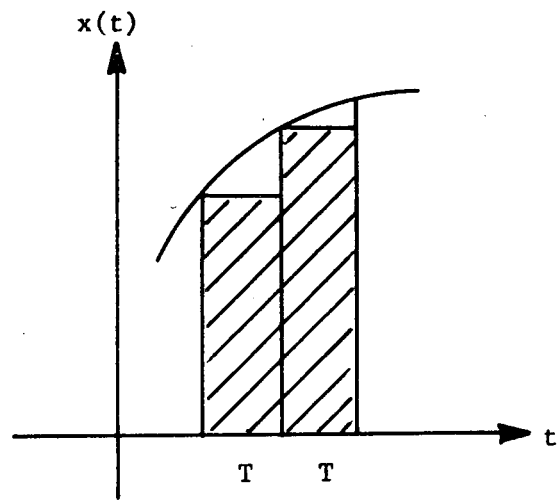
$$y(nT) = \int_0^{nT} x(t) dt . \quad (21)$$

Figure 12a illustrates the rectangular rule using the left side of the rectangles. Hence

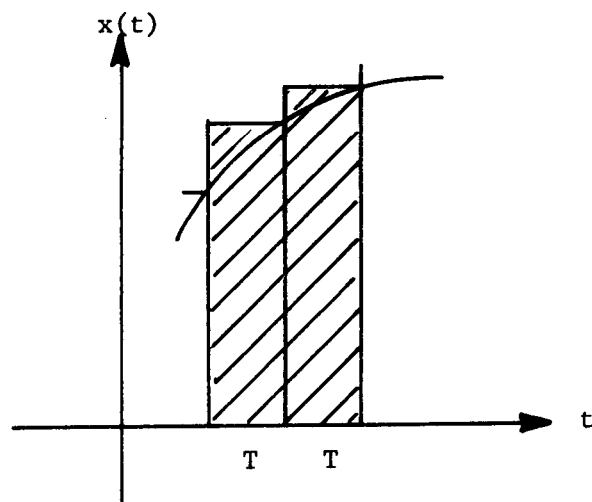
$$y(nT) = T \sum_{i=0}^{n-1} x(iT)$$

$$y(nT+T) = T \sum_{i=0}^n x(iT) = T \sum_{i=0}^{n-1} x(iT) + Tx(nT)$$

$$= y(nT) + Tx(nT)$$



(a) Left Side Rule



(b) Right Side Rule

Figure 12. The rectangular rule.

Therefore using equations (12) and (13)

$$\begin{aligned} D(z) &= \frac{Tz^{-1}}{1-z^{-1}} \\ &= \frac{T}{z-1} \end{aligned}$$

Hence we have approximated the integration transfer function

$$\frac{1}{s} \approx \frac{T}{z-1}$$

which gives the same results as equation (20) for the forward difference.

Right Side Rule. Figure 12b illustrates the use of the right side of the rectangle in approximating equation(21). Therefore

$$y(nT) = T \sum_{i=1}^n x(iT)$$

$$y(nT + T) = T \sum_{i=1}^{n+1} x(iT) = T \sum_{i=1}^n x(iT) + Tx(nT + T)$$

$$= y(nT) + T x(nT + T)$$

Letting $n = n - 1$

$$y(nT) = y(nT - T) + T x(nT)$$

Employing equations (12) and (13) one finds

$$D(z) = \frac{T}{1 - z^{-1}}$$

Hence, we have approximated the integrator

$$\frac{1}{s} \triangleq \frac{T}{1-z^{-1}}$$

which yields the identical result of equation (19) for the backward difference.

Trapezoidal Rule.

The trapezoidal rule takes the average of the left and right side of the rectangles in Figure 12. Hence

$$\begin{aligned} y(nT) &= \frac{T}{2} \sum_{i=0}^{n-1} [x(iT) + x(iT + T)] \\ &= \frac{1}{2} \left[T \sum_{i=0}^{n-1} x(iT) + T \sum_{i=1}^n x(iT) \right] \end{aligned}$$

Using the results of the rectangular rule,

$$\begin{aligned} D(z) &= \frac{1}{2} \left[\frac{Tz^{-1}}{1-z^{-1}} + \frac{T}{1-z^{-1}} \right] \\ &= \frac{T}{2} \frac{1+z^{-1}}{1-z^{-1}} \end{aligned}$$

Thus we have approximated

$$\frac{1}{s} \triangleq \frac{T}{2} \frac{1+z^{-1}}{1-z^{-1}}$$

This approximation is the familiar bilinear z - transform.

Simpson's Rule

Simpson's Rule evaluates equation (21) by the following formula

$$y(nT) = \frac{T}{3} [x(0) + 4x(T) + 2x(2T) + \dots + 4x(nT - T) + x(nT)]$$

But

$$y(nT + 2T) = y(nT) + \frac{T}{3} [x(nT) + 4x(nT + T) + x(nT + 2T)]$$

Letting $n = n - 2$ and following equations (12) and (13) yields

$$D(z) = \frac{T}{3} \frac{1 + 4z^{-1} + z^{-2}}{1 - z^{-2}}$$

Hence, we have approximated

$$\frac{1}{s} \approx \frac{T}{3} \frac{1 + 4z^{-1} + z^{-2}}{1 - z^{-2}}$$

Note that this formulation is valid only at even iterations (n even).

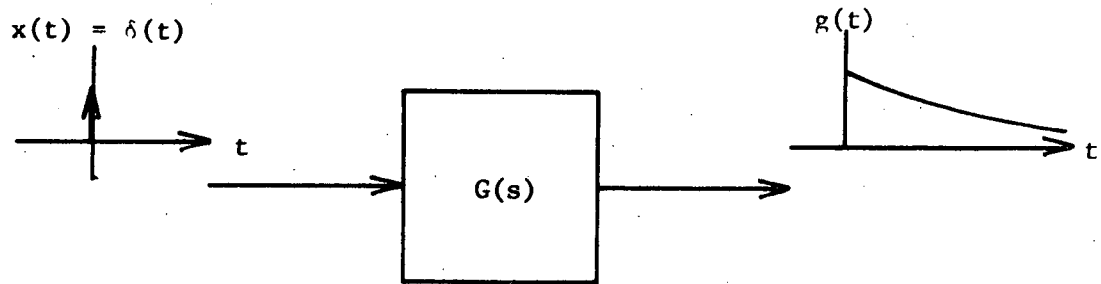
Impulse Invariance

Suppose that we want to find a discrete equivalent filter for the Laplace transfer function $G(s)$. Further suppose that we desire the impulse response of the discrete equivalent to match that of the analog filter as shown in Figure 13.

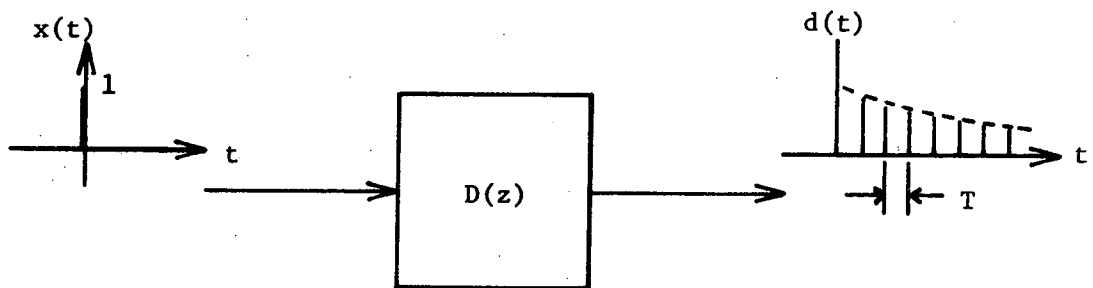
$$g(nT) = d(nT) .$$

Then

$$D(z) = \sum_{i=0}^{\infty} d(nT) z^{-n}$$



(a) Analog Filter



(b) Digital Filter

Figure 13. Impulse Invariance

$$\begin{aligned}
&= \sum_{i=0}^{\infty} g(nT) z^{-n} \\
&= G(z) ,
\end{aligned}$$

which is the standard z -transform. Hence, for impulse invariance

$$D(z) = Z[G(s)] = G(z)$$

the digital approximation is just the standard z -transform of $G(s)$.

Impulse Invariant Integrator

Let us find the digital equivalent of an analog integrator using impulse invariance and the models of Figure 14. We know that

$$G(z) = Z \left[\frac{1}{s} \right] = \frac{1}{1 - z^{-1}}$$

and that

$$G_{h_0}(s) = \frac{1 - e^{-Ts}}{s} \doteq T$$

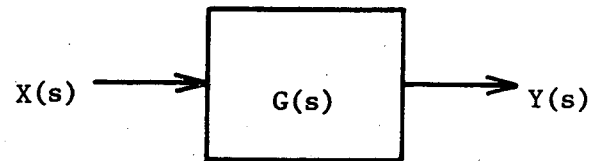
for small values of T . Hence

$$\frac{Y_d(z)}{X(z)} = \frac{T}{1 - z^{-1}}$$

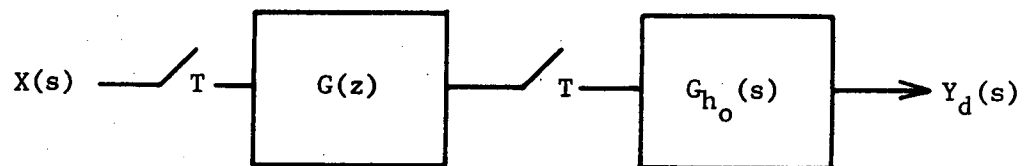
and we have again approximated

$$\frac{1}{s} \doteq \frac{T}{1 - z^{-1}}$$

Therefore, the backward difference, the right side rectangular rule, and the impulse invariant integrator all indicate equation (19) as their equivalent sampled-data transformation.



(a) Analog Integrator



(b) Digital Integrator

Figure 14. Impulse invariant integrator.

Mapping Functions Summary

As a result of our analysis of some elementary numerical approximation techniques we have identified several sampled data mapping functions.

Standard z-Transform

The standard z-transform yields an impulse invariant filter the mapping function for this transformation is

$$s = \frac{1}{T} \ln z \quad . \quad (22)$$

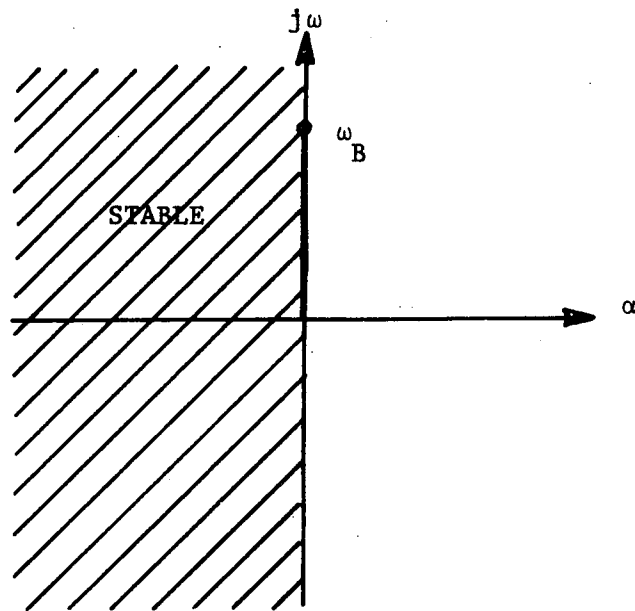
This mapping has been previously defined in Figure 9.

Backward Difference

The backward difference approximation for the solution of differential equations provides the following mapping

$$s = \frac{1 - z^{-1}}{T} \quad . \quad (23)$$

See Figure 15. Note that the region of stability in the s-plane maps into the right half plane $z^{-1} \geq 1$ of the z^{-1} plane. Since the region of instability in the z^{-1} plane is the interior of the unit circle, stable analog filters will always result in stable digital equivalents. In fact some unstable analog filters give stable digital ones. A major disadvantage of this mapping is seen in the frequency response contour. The $j\omega$ axis in the s-plane does not map to the unit circle in the z^{-1} plane (or the z-plane either). Hence, as we get farther from $s = 0$ or $z = 1$ the more degraded will be our desired frequency response. Thus,



(a) s-plane

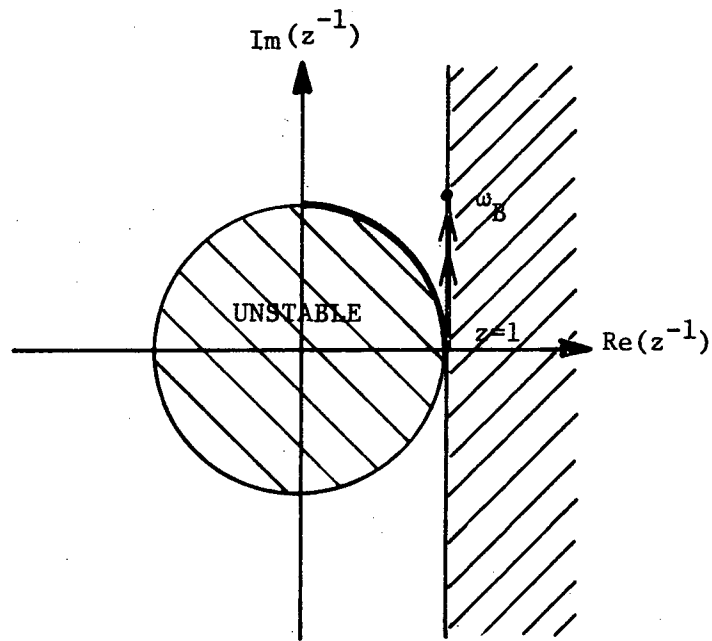
(b) z^{-1} plane

Figure 15. Mapping $s = \frac{1 - z^{-1}}{T}$

we must decrease T (increase f_s) to improve this approximation.

Forward Difference

The forward difference approximation suggested the following mapping

$$s = \frac{z - 1}{T} . \quad (24)$$

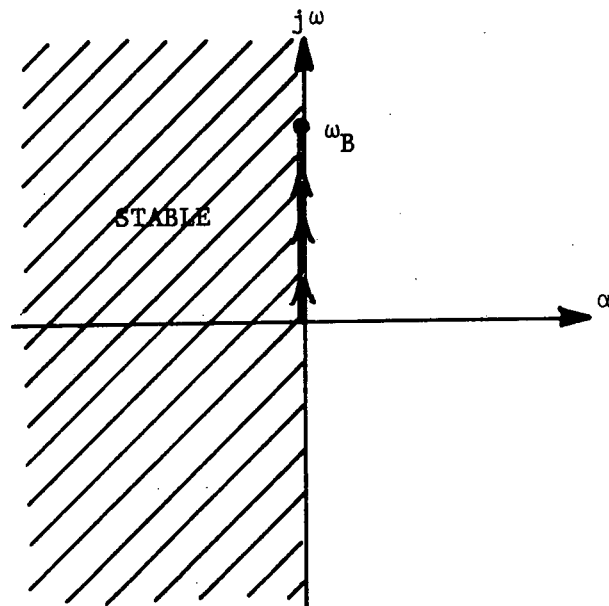
This mapping function is shown in Figure 16. Note that the left-half plane in the s domain maps to the region to the left of $z = 1$ in the z -plane. But the interior of the unit circle represents the stability region in the z -plane. Consequently, some stable analog filters will give unstable digital ones. Unstable analog filters will also be unstable digital ones under this mapping. Yet a further disadvantage is the same frequency contour encountered in Figure 15. Hence, this is an undesirable mapping.

Bilinear z-Transformation

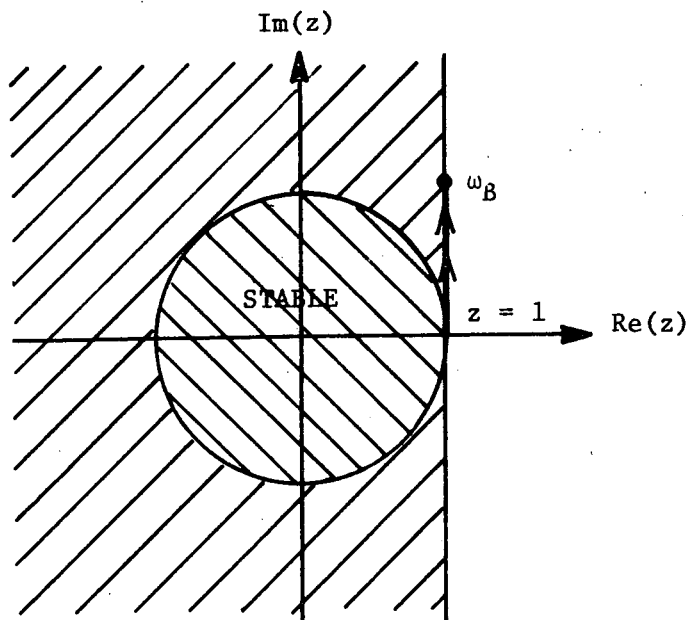
The trapezoidal integration approximation led to the sampled data mapping

$$s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} . \quad (25)$$

This mapping is illustrated in Figure 17. Notice here that the entire left-half s -plane maps to the interior of the unit circle in the z -plane. Hence, all stable analog filters will result in stable digital ones. Also, the $j\omega$ axis in the s -plane maps to the unit circle in the z -plane. However, the entire $j\omega$ axis maps onto the unit circle which causes a mismatching of frequencies. This is a direct result of the

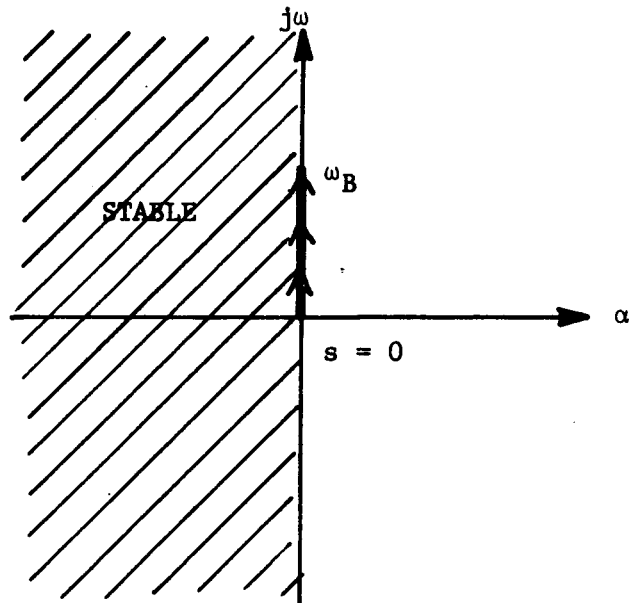
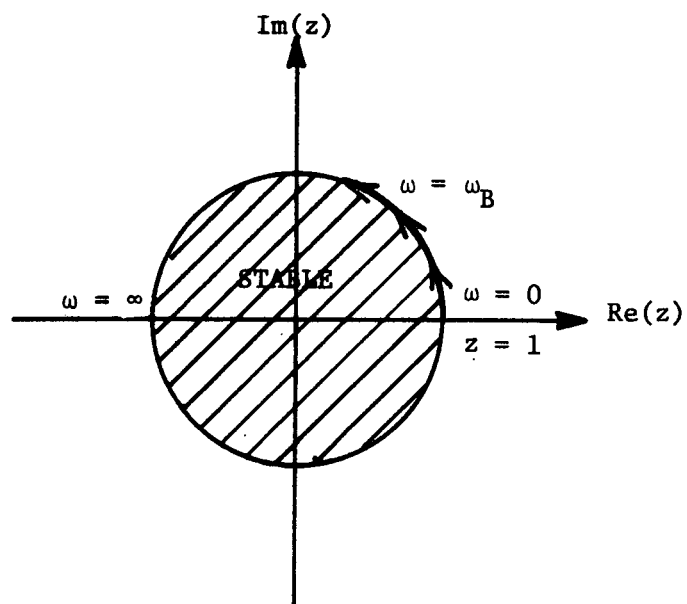


(a) s-plane



(b) z-plane

Figure 16. Mapping $s = \frac{z - 1}{T}$.

(a) s -plane(b) z -planeFigure 17. Bilinear z -transform.

characteristic that for a digital filter

$$z = 1 \rightarrow \omega = 0$$

$$z = j1 \rightarrow \omega = \omega_s/4$$

$$z = -1 \rightarrow \omega = \omega_s/2$$

as required by equation (18). For the bilinear z-transform the frequencies in the z-plane (ω_D) are related to frequencies in the s-plane (ω_A) by

$$j\omega_A = \frac{e^{j\omega_D T} - 1}{e^{j\omega_D T} + 1} = \frac{2j \sin \frac{\omega_D T}{2}}{2 \cos \frac{\omega_D T}{2}}$$

or

$$\omega_D = \frac{2}{T} \tan^{-1} \omega_A \quad (26)$$

See Figure 18. Correction for this frequency scale warping may be accomplished by redesigning (prewarping) the critical frequencies of the desired transfer function $G(s)$ before applying the bilinear z-transform.

This transformation maps circles and straight lines in the s-plane to circles in the z-plane. It works well for frequency characteristics which are piecewise linear. It also insures that no frequency aliasing can occur in the transfer function frequency characteristic because the $+j\omega$ axis does map into the upper half of the unit circle. Hence, the bilinear z-transform is quite popular.

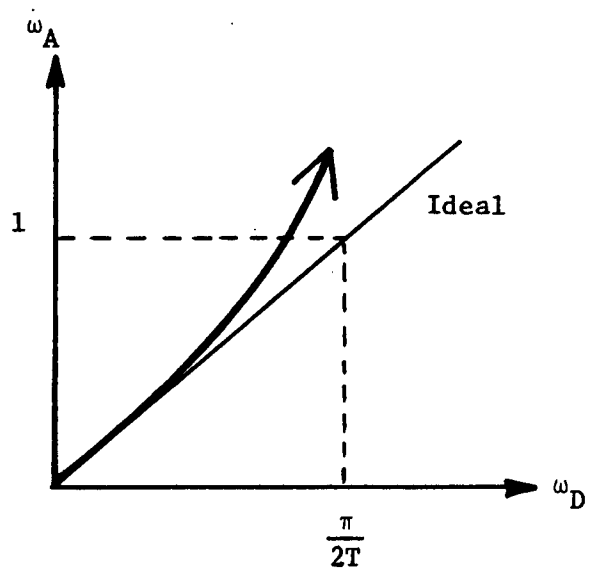


Figure 18. Change in frequency scale for bilinear z-transform.

Matched z-Transforms [2]

The standard z-transform of equations (3b) and (4) required a partial fraction expansion of $G(s)$ in order to complete the mapping

$$\frac{1}{s + u} = \frac{1}{1 - e^{-uT} z^{-1}}$$

For the purpose of simplifying the calculations, the matched z-transform maps the poles and zeroes ($-b_j$ and $-a_j$ of equation (3a)) to the z-plane as follows:

$$s + \alpha \rightarrow 1 - z^{-1} e^{-\alpha T} \quad (27)$$

Hence the matched z-transform of equation (3a) is

$$\begin{aligned} G(z) = G(s) & \left| \begin{array}{l} s + a_i = 1 - z^{-1} e^{-a_i T} \\ s + b_j = 1 - z^{-1} e^{-b_j T} \end{array} \right. \\ & \prod_{i=1}^m (1 - z^{-1} e^{-a_i T}) \\ & = K \frac{1 = 1}{\prod_{j=1}^n (1 - z^{-1} e^{-b_j T})} \end{aligned} \quad (28)$$

where K is adjusted to give the desired gain at d.c. ($z = 1$). This transform matches the poles and zeroes in the s and z planes. Note that the poles of this transform are identical with those of the standard z-transform but that the zeroes are different. Because of this difference, the matched z-transform may be used on nonbandlimited inputs. If $G(s)$ has no zeroes, it is sometimes necessary to multiply $(1 + z^{-1})^N$, N an integer, times the expression (28).

Other Transforms

In general any transformation which maps the stable region of the s-plane into the stable region of the z-plane may be used. It is helpful for the $j\omega$ axis in the s-plane to map to the z-plane's unit circle. Another important property is that rational functions $G(s)$ should be transformed into rational functions $D(z)$ so that the proper difference equations may be determined for realization.

Simpson's Rule

The Simpson's Rule approximation suggested that the mapping

$$s = \frac{3}{T} \frac{1 - z^{-2}}{1 + 4z^{-1} + z^{-2}} \quad (29)$$

be used as a transformation. The analysis of this mapping is left as an exercise for the reader. Please note that a second-order function $G(s)$ will transform to a fourth-order $D(z)$. This is undesirable from a digital hardware viewpoint.

(w,v)-Transform [14]

In some applications, the system transfer function $G(s, z, z^\alpha)$ may be a function of s , $z = e^{Ts}$, and z^α , where $0 < \alpha < 1$. If all initial conditions are zero and

$$w = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}}$$

$$v(\alpha) = 1 - \alpha(1 - z^{-1}) + \frac{\alpha(\alpha - 1)}{2} (1 - z^{-1})^2,$$

then for a system described by

$$Y(s) = G(s, z, z^\alpha) X(s)$$

its z-transform will be

$$Y(z) = G(w, z, v(\alpha)) \left[X(z) - \frac{x(0)}{1 + z^{-1}} \right] .$$

If $x(0) = 0$, then

$$D(z) = G(s, z, z^\alpha) \left| \begin{array}{l} s = w \\ z^\alpha = v(\alpha) \end{array} \right. .$$

This completes the definition of the (w, v) transform.

Example. Scott [15] has shown that a desirable phase lock loop has the transfer function

$$G(s) = \frac{10}{s + 10z^{-0.5}}$$

Using the (w, v) transform to find a digital equivalent if $x(0) = 0$

$$D(z) = \frac{10}{s + 10z^{-0.5}} \left| \begin{array}{l} s = w = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \\ z^{-0.5} = v(0.5) \end{array} \right.$$

$$v(0.5) = 1 - 0.5(1 - z^{-1}) + \frac{0.5(-0.5)}{2} (1 - z^{-1})^2$$

$$= 0.375 + .75z^{-1} - .125z^{-2}$$

$$D(z) = \frac{5T(1 + z^{-1})}{(1 + 1.875T) - (1 - 5.625T)z^{-1} + (3.125T)z^{-2} - (.625T)z^{-2}}$$

IV. DISCRETE STATE VARIABLES [5]

An n^{th} order discrete-time system is generally described by difference equations. The difference equation description of the system dynamics may be alternately presented in vector matrix (state variable) form by the following set of first-order difference equations.

$$\begin{aligned}\underline{x}(kT + T) &= A\underline{x}(kT) + B\underline{u}(kT) \\ \underline{y}(kT) &= C\underline{x}(kT) + D\underline{u}(kT),\end{aligned}\tag{30}$$

where $\underline{x}(kT)$, $\underline{u}(kT)$, and $\underline{y}(kT)$ are vectors of the discrete state variables, input variables, and output variables respectively. The symbol T is the sampling period and k is any non-negative integer.

For the purpose of simplifying the notation, the sampling period T shall hereafter be omitted from the equation; thus, equation (30) becomes

$$\begin{aligned}\underline{x}(k + 1) &= A\underline{x}(k) + B\underline{u}(k) \\ \underline{y}(k) &= C\underline{x}(k) + D\underline{u}(k)\end{aligned}\tag{31}$$

The solution of equation (30) can be found by rewriting the first of equations (31) in standard z-transform notation:

$$z \underline{X}(z) = A\underline{X}(z) + B\underline{U}(z) + z\underline{x}(0),$$

where $\underline{x}(0)$ is a vector of the initial condition of the state variables. Solving for $\underline{X}(z)$ produces

$$\underline{X}(z) = (zI - A)^{-1}z\underline{x}(0) + (zI - A)^{-1}B\underline{U}(z).\tag{32}$$

The inverse z-transform (Z^{-1}) of $(zI - A)^{-1}z$ is

$$Z^{-1}[(zI - A)^{-1}z] = \phi(k) = A^k$$

Therefore, the inverse z-transform of (32) yields

$$\underline{x}(k) = A^k \underline{x}(0) + \sum_{n=0}^{k-1} A^{k-1-n} B \underline{u}(n) . \quad (33)$$

This solution demonstrates that the present state of the system $\underline{x}(k)$ is dependent upon the initial state $\underline{x}(0)$ and the system inputs $\underline{u}(n)$ from the initial time ($t = 0$) to the present time ($t = kT$).

V. CONVOLUTION

In this section a review of continuous and discrete linear systems is presented. The equations are discussed in rapid succession.

Continuous Linear Systems

Figure 19a illustrates the conventional continuous system under consideration. Any linear system obeys superposition and is characterized by the impulse response $g(t, \xi)$, the response at time t due to an impulse at ξ . Hence

$$y(t) = \int_{-\infty}^{\infty} x(\xi)g(t, \xi)d\xi, \quad (34)$$

which is called the superposition integral. If the linear system is shift invariant then

$$g(t, \xi) = g(t - \xi)$$

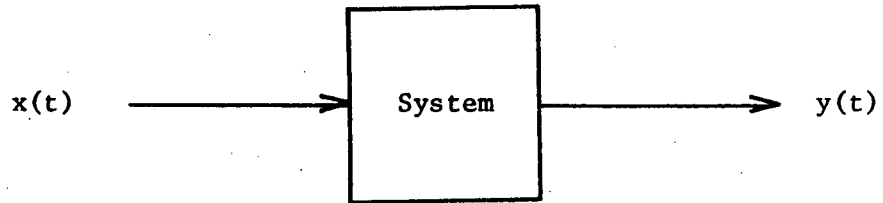
and equation (34) becomes

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} x(\xi)g(t - \xi)d\xi, \\ &\triangleq x(t)*g(t) \end{aligned} \quad (35)$$

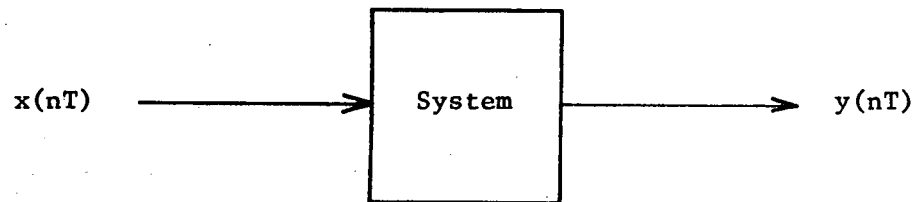
which is the convolution integral.

But, by definition of the Laplace transform

$$Y(s) = \int_0^{\infty} y(t)e^{-st}dt.$$



(a) Continuous System



(b) Discrete System

Figure 19. Convolution.

Taking the Laplace transform of equation (35) produces

$$Y(s) = G(s)X(s),$$

where

$$G(s) = \int_0^{\infty} g(t)e^{-st} dt$$

is called the system transfer function;

and $G(j\omega)$, the frequency response of the system.

Discrete Linear Systems

Figure 19b illustrates the conventional discrete-time system under consideration. The linear discrete system also obeys superposition and is also characterized by the impulse response $d(nT, kT)$, the response at time nT due to and discrete impulse $\delta(kT)$, where

$$\begin{aligned} \delta(kT) &= 1 & \text{if } k &= 0 \\ &= 0 & \text{if } k &\neq 0 \end{aligned} \tag{36}$$

and

$$x(nT) = \sum_{k=0}^{\infty} x(kT)\delta(nT - kT)$$

By superposition

$$y(nT) = \sum_{k=0}^{\infty} x(kT)d(nT, kT), \tag{37}$$

which is called the superposition sum. If the system is shift invariant, then

$$d(nT, kT) = d(nT - kT)$$

and equation (37) becomes

$$y(nT) = \sum_{k=0}^{\infty} x(kT)d(nT - kT), \quad (38)$$

$$\triangleq x(nT) * d(nT)$$

the convolution sum. But by definition of the standard z-transform

$$Y(z) = \sum_{k=0}^{\infty} y(kT)z^{-k}.$$

Taking the z-transform of equation (38) produces

$$Y(z) = D(z)X(z)$$

which is the identical result in equation (11). Hence, $D(z)$ is called the discrete system transfer function and $D(e^{j\omega T})$ is called the frequency response (see equation (18)).

VI. DISCRETE FOURIER TRANSFORM

This section examines the properties of continuous Fourier transforms and derives the discrete approximation.

Continuous Fourier Transform

The Fourier transformation may be defined by

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-j2\pi ft} dt \quad (39)$$

and the inverse Fourier transform as

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{j2\pi ft} df . \quad (40)$$

The similarity between equations (39) and (40) is illustrated by the summary of Fourier transform pairs listed in Table 1.

Another useful property of Fourier transforms is shown below:

$$\int_{-\infty}^{\infty} |g(t)|^2 dt = \int_{-\infty}^{\infty} |G(f)|^2 df. \quad (41)$$

This is known as the Fourier Integral Energy Theorem.

Discrete Fourier Transform [6]

Sampling Process

In Figure 4, an impulse sampler was presented which sampled a signal for $t \geq 0$. However, if the signal is zero for negative t , the following sampling function produces the same effect:

TABLE 1. Fourier Transform Pairs

Time Function $g(t)$	Fourier Transform $G(f)$
$\delta(t)$	1
1	$\delta(t)$
$u(t + \frac{A}{2}) - u(t - \frac{A}{2})$	$A \frac{\sin \pi A f}{\pi A f}$
$A \frac{\sin \pi A t}{\pi A t}$	$u(f + \frac{A}{2}) - u(f - \frac{A}{2})$
$\delta(t + \frac{A}{2}) - \delta(t - \frac{A}{2})$	$2 \cos \pi A f$
$2 \cos \pi A t$	$\delta(f + \frac{A}{2}) + \delta(f - \frac{A}{2})$
$\frac{d}{dt} g(t)$	$j 2 \pi f G(f)$
$-j 2 \pi t g(t)$	$\frac{d}{df} G(f)$
$\int_0^t g(x) dx$	$\frac{G(f)}{j 2 \pi f}$
$g(At)$	$\frac{1}{A} G(\frac{f}{A})$
$A g_1(t) + B g_2(t)$	$A G_1(f) + B G_2(f)$
$g(t + A)$	$e^{j 2 \pi f A} G(f)$
$e^{2 \pi A t} g(t)$	$G(f + j A)$
$g(t) * x(t)$	$G(f) X(f)$
$g(t) x(t)$	$G(f) * X(f)$
$\sum_{k=-\infty}^{\infty} \delta(t - kT)$	$\frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(f - \frac{k}{T})$

$$\Delta(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT). \quad (42)$$

From Table 1 the Fourier transform of $\Delta(t)$ is

$$F[\Delta(t)] = \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta\left(f - \frac{k}{T}\right).$$

In order to verify this result we may note that $F[\Delta(t)]$ is periodic and may be expressed in a Fourier Series as

$$F[\Delta(t)] = \sum_{n=-\infty}^{\infty} c_n e^{-j2\pi n T f}$$

where

$$c_n = T \int_{-\frac{1}{2T}}^{\frac{1}{2T}} F[\Delta(t)] e^{j2\pi n T f} df$$

$$c_n = T \int_{-\frac{1}{2T}}^{\frac{1}{2T}} \frac{1}{T} df = 1.$$

Hence

$$F[\Delta(t)] = \sum_{n=-\infty}^{\infty} e^{-j2\pi n T f},$$

as expected from equation (42).

Suppose the sampling function in equation (42) is multiplied by an input signal $g(t)$ to produce the sampled signal

$$\begin{aligned}
 g^*(t) &= g(t)\Delta(t) \\
 &= \sum_{k=-\infty}^{\infty} g(kT)\delta(t - kT).
 \end{aligned}$$

As illustrated in Figure 5, if $B/2\pi$, the highest frequency in the sampled signal is less than $f_s/2$, then recovery of the original signal is possible with an ideal low-pass filter whose cutoff is f_s . To recover the signal, $g_r(t)$ we multiply $G^*(f)$ by a square window function $F_{lp}(f)$.

$$G_r(f) = G^*(f)F_{lp}(f) \quad (43)$$

Since multiplication of Fourier transforms represents convolution in the time domain

$$\begin{aligned}
 g_r(t) &= g^*(t) * \frac{\sin(\pi t/T)}{\pi t/T} \\
 &= \left[\sum_{k=-\infty}^{\infty} g(kT)\delta(t - kT) \right] * \left[\frac{\sin(\pi t/T)}{\pi t/T} \right] \\
 g_r(t) &= \sum_{k=-\infty}^{\infty} g(kT) \frac{\sin \frac{\pi}{T}(t - kT)}{\frac{\pi}{T}(t - kT)}. \quad (44)
 \end{aligned}$$

If $f_s > B/\pi$, the low pass filter is ideal, and the samples $g(kT)$ are exact, then

$$g_r(t) = g(t) .$$

However, the samples are never exact, no signal is ever bandlimited, and no low-pass filter is ideal. Therefore, we can't exactly recover a sampled signal.

DFT Derivation

Now discrete versions for (39) and (40) will be determined. Define the following conditions for (39):

f_s = sampling frequency

$T = 1/f_s$ = sampling interval

$g(t) = 0$ outside the interval $t = [0, NT]$

N = an integer, the number of sample points

$G(f)$ is bandlimited to $\pm f_s/2$.

Please note that these conditions can never be completely satisfied. With the time-limited function $g(t)$, $G(f)$ cannot be bandlimited. In practice it can get quite small as $|f|$ increases. However, since a function is never time and bandlimited, the time and frequency samples are corrupted by aliasing.

Using the above conditions in equation (39) one obtains

$$G(f) = \int_0^{NT} g(t) e^{-j2\pi ft} dt .$$

Using the rectangular rule for numerical integration

$$G(f) = T \sum_{k=0}^{N-1} g(kT) e^{-j2\pi fkT} . \quad (45)$$

The discrete version of equation (39) will compute samples of $G(f)$ every $\Delta f = f_s/N = 1/NT$ Hertz. Substituting $f = n\Delta f$ into equation (45)

$$G(n\Delta f) = T \sum_{k=0}^{N-1} g(kT) e^{-j2\pi n\Delta f kT}$$

or

$$G(n/NT) = T \sum_{k=0}^{N-1} g(kT) e^{-j(2\pi/N)nk}$$

In another form

$$G(n/NT) = T \sum_{k=0}^{N-1} g(kT) W^{-nk} \quad (46)$$

where

$$W = e^{j2\pi/N}$$

Since $g(t) = 0$ outside the interval $0 < t < NT$, one can construct a periodic function $h(t)$ from $g(t)$, with period NT :

$$h(t) = \sum_{m=-\infty}^{\infty} g(t - mNT), \quad (47)$$

which may be written

$$\begin{aligned} h(t) &= \int_0^t g(\tau) \sum_{m=-\infty}^{\infty} \delta(t - mNT - \tau) d\tau \\ &= g(t) * \sum_{m=-\infty}^{\infty} \delta(t - mNT) \end{aligned}$$

where the $*$ denotes convolution. Since $G(f)$ is bandlimited to $f_s/2$, so is $H(f)$. Therefore,

$$\begin{aligned} H(f) &= G(f) \left[\frac{1}{NT} \sum_{m=-\infty}^{\infty} \delta(f - m/NT) \right] \\ &= \sum_{m=-\infty}^{\infty} \left[\frac{G(m/NT)}{NT} \right] \delta(f - m/NT), \end{aligned}$$

and

$$H(f) = \sum_{m=-\infty}^{\infty} H'(m/NT) \delta(f - m/NT), \quad (48)$$

where $H(f)$ is the continuous Fourier transform of $h(t)$. The weighting function $H'(m/NT)$ in equation (48) is defined below:

$$H'(m/NT) = (1/NT) G(m/NT). \quad (49)$$

Equation (46) may be inserted in (49),

$$H'(m/NT) = (1/N) \sum_{k=0}^{N-1} g(kT) W^{-mk} \quad (50)$$

From (47),

$$g(kT) = h(kT): \quad k=0, N-1.$$

Therefore (50) becomes

$$H'(m/NT) = (1/N) \sum_{k=0}^{N-1} h(kT) W^{-mk} \quad (51)$$

IDFT Derivation

The inverse discrete Fourier transform is found by considering equation (40)

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{j2\pi ft} df.$$

Under the conditions of the previous section

$$h(t) = \int_{-1/2T}^{1/2T} H(f) e^{j2\pi ft} df \quad (52)$$

since $H(f)$ is bandlimited to $f_s/2 = 1/2T$.

Substituting equation (48) into (52) and evaluating at $t = kT$,

$$h(kT) = \int_{-1/2T}^{1/2T} \sum_{m=-\infty}^{\infty} H'(m/NT) e^{j2\pi f kT} \delta(f - m/NT) df$$

Since the integrand is periodic $(1/T)$ in f ,

$$h(kT) = \int_0^{1/T} \sum_{m=-\infty}^{\infty} H'(m/NT) e^{j(2\pi/N)mk} \delta(f - m/NT) df$$

The limits of integration truncate the sum to

$$h(kT) = \sum_{m=0}^{N-1} H'(m/NT) W^{mk}. \quad (53)$$

The prime is dropped in equations (51) and (53) for convenience, and the resulting relations are

DFT

$$H(m/NT) = (1/N) \sum_{k=0}^{N-1} h(kT) W^{-mk} \quad (54)$$

IDFT

$$h(kT) = \sum_{m=0}^{N-1} H(m/NT) W^{mk} \quad (55)$$

These equations define the discrete Fourier transform pair. This transform may be thought of as a mapping of N points in the time domain to N points in the frequency domain.

DFT Pairs

Although equations (54) and (55) are discrete approximations of (39) and (40), we can show that they form exact transform pairs.

From equation (54),

$$\underline{H} = \frac{1}{N} [W] \underline{h} \quad (56)$$

where \underline{H} and \underline{h} are vectors constructed of the N samples in the frequency and time domains, and

$$[W] = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & W^{-1} & \dots & W^{-2(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & W^{-(N-1)} & \dots & W^{-(N-1)^2} \end{bmatrix} \quad (57)$$

From equation (55)

$$\underline{h} = [W^*] \underline{H} \quad (58)$$

where $[W^*]$ is the complex conjugate of $[W]$. In order to prove that the transform pairs are exact one must show that

$$[W]^{-1} = \frac{1}{N} [W^*] \quad (59)$$

This proof follows:

Let

$$[P] = [W][W^*]$$

then a general element of $[P]$ is

$$P_{\ell m} = [1 \ W^{-(\ell-1)} \ \dots \ W^{-(N-1)(\ell-1)}]$$

$$\begin{bmatrix} 1 \\ W^{(m-1)} \\ \vdots \\ W^{(N-1)(m-1)} \end{bmatrix}$$

$$= 1 + W^{-(\ell-m)} + \dots + W^{-(N-1)(\ell-m)}$$

Then all diagonal elements of $[P]$

$$P_{\ell\ell} = 1 + 1 + \dots + 1 = N$$

and the off diagonal elements

$$P_{\ell m} = \frac{1 - W^{N(m-\ell)}}{1 - W} = 0.$$

Hence

$$[P] = N[I]$$

and the exact relationship is proved.

A summary of DFT pairs is listed in Table 2. Several other interesting relations are displayed below:

$$h(0) = \sum_{m=0}^{N-1} H(m/NT) \quad (60)$$

$$H(0) = (1/N) \sum_{k=0}^{N-1} h(kT) \quad (61)$$

and

$$\frac{1}{N} \sum_{k=0}^{N-1} |h(kT)|^2 = \sum_{m=0}^{N-1} |H(m/NT)|^2. \quad (62)$$

This last relation is known as Parseval's Theorem.

Fast Fourier Transform

Calculation Time

The fast Fourier transform (FFT) is a high speed technique for calculating the DFT. If the number of samples N may be written

$$N = r_1 r_2 \dots r_n, \quad r_i \text{ an integer}$$

then

$$[W] = [W_1][W_2] \dots [W_n] \quad (63)$$

where $[W_i]$ is an $N \times N$ matrix with only $r_i N$ non-zero elements. The calculation of

$$\underline{H} = \frac{1}{N} [W] \underline{h}$$

TABLE 2. DFT Pairs

$h(kT)$	$H(m/NT)$
$Ah_1(kT) + Bh_2(kT)$	$AH(m/NT) + BH(m/NT)$
$h(kT - nT)$	$W^{-mn}H(m/NT)$
$h_1(kT)h_2(kT)$	$H_1(m/NT)*H_2(m/NT)$
$h^*(kT)$	$H^*(-m/NT)$
$h(-kT)$	$H(-m/NT)$
$\delta(kT)$	$1/N$
$\delta(kT - nT)$	$(1/N)W^{-mn}$
$\frac{1}{N} \sum_{\ell=0}^{N-1} h_1(\ell + k)h_2(\ell)$	$H_1(m/NT)H_2(-m/NT)$

requires N^2 operations of complex multiplication, whereas

$$\underline{H} = \frac{1}{N} [W_1][W_2] \cdots \underbrace{[W_n]}_{r_n N \text{ operations}} \underline{h} \quad (64)$$

requires $(r_1 + r_2 + \cdots + r_n) N$ operations. For the special case $r_1 = 2$, $N = 2^n$, the total number of operations is

$$\begin{aligned} \#oper &= (2 + 2 + \cdots + 2)N \\ &= 2nN \\ &= 2N \log_2 N \end{aligned} \quad (65)$$

Example. Compare the time to calculate the DFT and FFT of a sequence of 1024 samples of a time function given that a typical computer calculates a complex multiplication in about $40\mu s$

DFT:

$$\begin{aligned} \text{Calc. Time} &\doteq N^2(40\mu s) \\ &= 1.053 \times 10^6 \times 40 \times 10^{-6} \\ &= 42.1 \text{ seconds} \end{aligned}$$

FFT:

$$\begin{aligned} \text{Calc. Time} &\doteq 2N \log_2 N(40\mu s) \\ &= (2048)(10)(40 \times 10^{-6}) \\ &= .82 \text{ seconds} \end{aligned}$$

FFT Derivation

Since the DFT is a linear operation and $N = 2^n$, we may break equation (54) into two functions, the even samples and the odd ones:

$$\begin{aligned} H(m/NT) &= \frac{1}{N} \sum_{k=0}^{(N/2)-1} [h(2kT)W^{-2mk} + h(2kT+T)W^{-2mk}W^{-m}] \\ &= \frac{1}{N} \sum_{k=0}^{(N/2)-1} h(2kT)W^{-2mk} + \frac{W^{-m}}{N} \sum_{k=0}^{(N/2)-1} h(2kT+T)W^{-2mk} \end{aligned}$$

or

$$H(m/NT) = \text{DFT}[h(2kT)] + W^{-m} \text{DFT}[h(2kT+T)] \quad (66)$$

for $m = 0, \frac{N}{2} - 1$. But

$$W^{-m} - \frac{N}{2} = W^{-m}W^{-\frac{N}{2}} = -W^{-m}$$

$$W^{-2(m + \frac{N}{2})} = W^{-2m}W^{-N} = W^{-2m}.$$

Therefore, the remaining samples may be determined by

$$\begin{aligned} H(m/NT + 1/2T) &= \frac{1}{N} \sum_{k=0}^{(N/2)-1} h(2kT)W^{-2mk} \\ &\quad - \frac{W^{-m}}{N} \sum_{k=0}^{(N/2)-1} h(2kT+T)W^{-2mk} \end{aligned}$$

or

$$H(m/NT + 1/2T) = \text{DFT}[h(2kT)] - W^{-m} \text{DFT}[h(2kT+T)] \quad (67)$$

for $m = 0, \frac{N}{2} - 1$. The above equations may be successively applied in order to achieve the maximum reduction in computation time indicated by equation (65). The technique of dividing the time samples into even and odd parts is sometimes called "decimation in time." The FFT of eight-points is illustrated in Figure 20.

Note that the time samples are entered in "bit reverse" order:

	binary	bit reversal	
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

IFFT Derivation

Since the IDFT is a linear operation and $N = 2^n$, we may separate equation (55) into two functions, the even samples and the odd ones:

$$\begin{aligned}
 h(kT) &= \sum_{m=0}^{(N/2)-1} [H(2m/NT)W^{2mk} + H(2m + 1/NT)W^{2mk}W^k] \\
 &= \sum_{m=0}^{(N/2)-1} H(2m/NT)W^{2mk} + W^k \sum_{m=0}^{(N/2)-1} H(2m + 1/NT)W^{2mk}
 \end{aligned}$$

or

$$h(kT) = \text{IDFT}[H(2m/NT)] + W^k \text{IDFT}[H(2m + 1/NT)], \quad (68)$$

for $k = 0, \frac{N}{2} - 1$. But

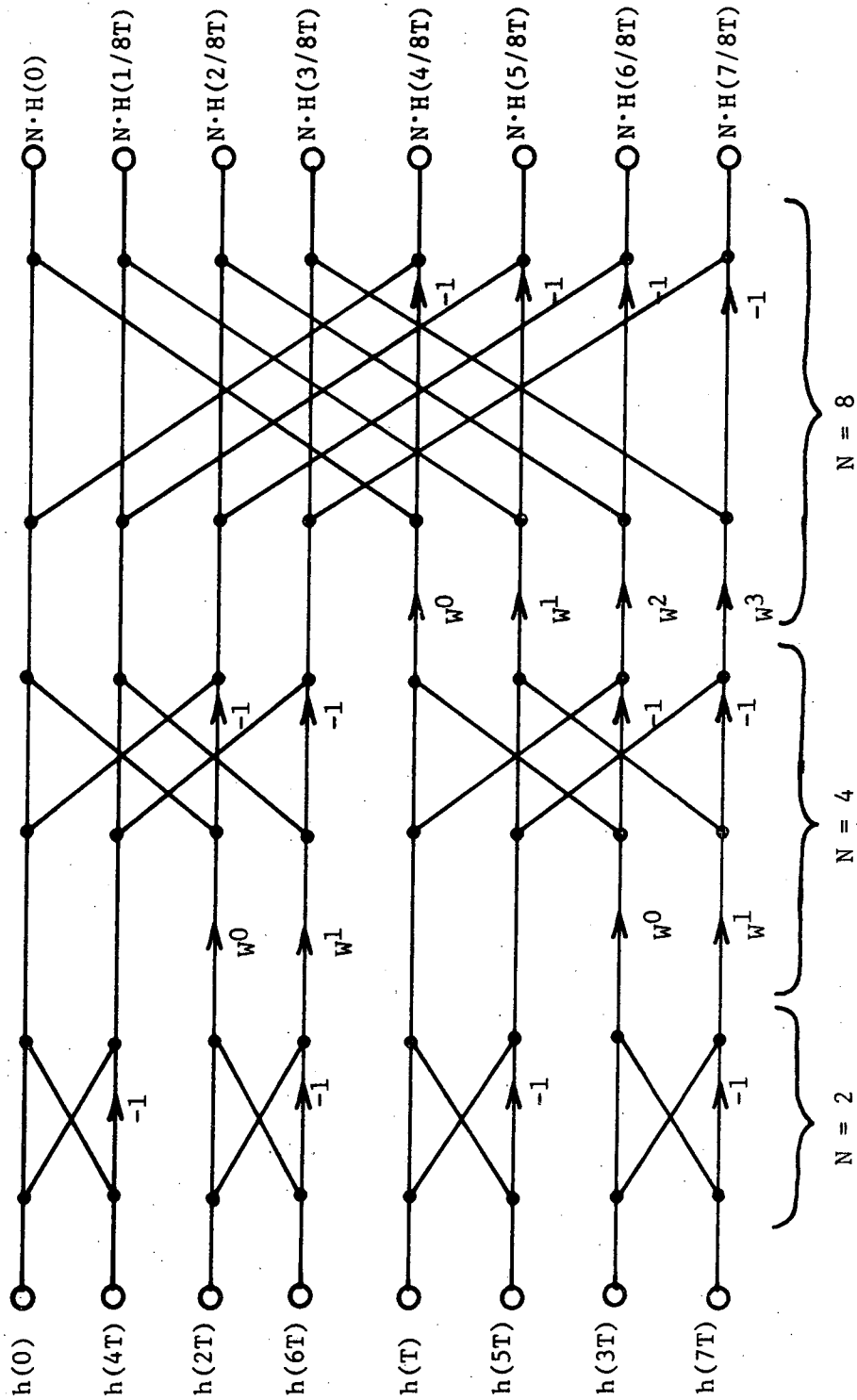


Figure 20. FFT for eight samples.

$$W^{k + \frac{N}{2}} = W^k W^{\frac{N}{2}} = -W^k$$

$$W^{2(k + \frac{N}{2})} = W^{2k} W^N = W^{2k} .$$

Therefore the remaining samples may be calculated by

$$h(kT + \frac{NT}{2}) = \sum_{m=0}^{(N/2)-1} H(2m/NT) W^{2mk} - W^k \sum_{m=0}^{(N/2)-1} H(2m + 1/NT) W^{2mk}$$

or

$$h(kT + \frac{NT}{2}) = \text{IDFT}[H(2m/NT)] - W^k \text{IDFT}[H(2m + 1/NT)] \quad (69)$$

for $k = 0, \frac{N}{2} - 1$. The repetition of this process yields the algorithm for the IFFT. The above derivation is sometimes called "decimation in frequency."

Equations (66) through (69) suggest an algorithm for calculating the IFFT as shown in Figure 21. In this figure the equations (68) and (69) are employed at each stage of the transformation of eight frequency samples into eight time samples. Note that the frequency samples are again inserted in "bit reversals" order.

The reader will please note the similarity of the Figures 20 and 21. The basic element is sometimes called a "butterfly" as shown in Figure 22. The gains on a few multipliers are different. This structure suggests the mechanization of FFT hardware.

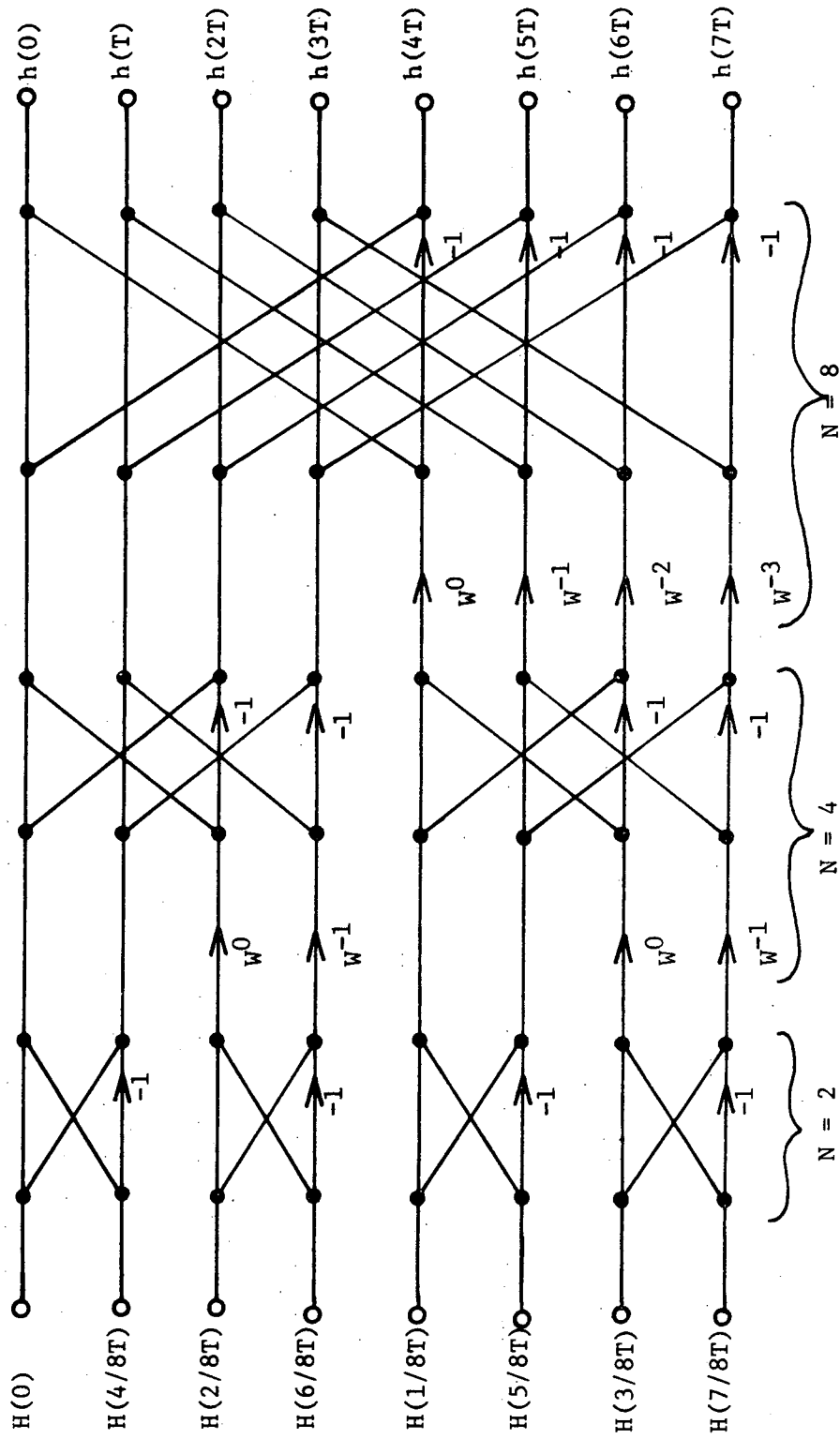


Figure 21. IFFT for eight samples.

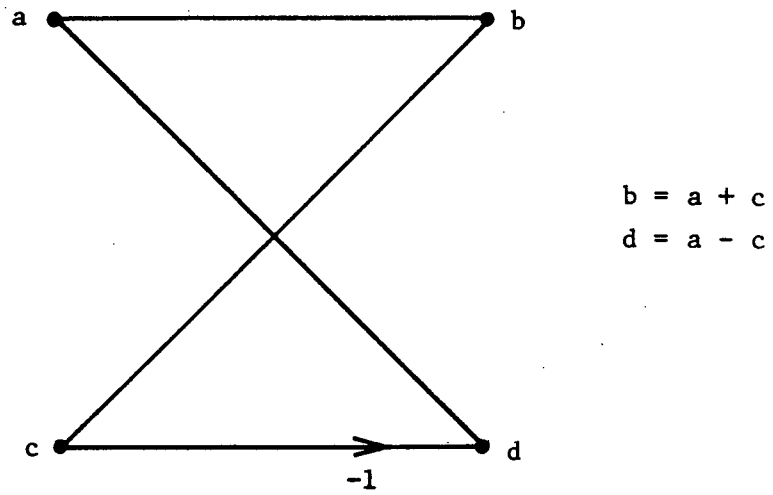


Figure 22. Butterfly structure.

VII. RANDOM PROCESSES

In the analysis and synthesis of digital filters one frequently encounters signals which are random in nature that must be examined with special techniques.

Continuous Processes [6]

If $G(f)$ is the Fourier transform of a continuous signal $g(t)$, the power density function or power spectrum may be defined as

$$\Psi_{gg}(f) = \lim_{A \rightarrow \infty} \left\{ \frac{1}{A} |G(f)|^2 \right\}.$$

The auto correlation function

$$\psi_{gg}(\tau) = \lim_{A \rightarrow \infty} \frac{1}{A} \int_0^A g(t)g(t + \tau)dt.$$

These two functions form a Fourier transform pair

$$\Psi_{gg}(f) = \int_{-\infty}^{\infty} \psi_{gg}(\tau) e^{j2\pi f\tau} d\tau$$

Both the power spectrum and the auto correlation function are real and symmetric.

Discrete Processes [16]

For the discrete case, the cross-correlation function is first defined

$$\psi_{xy}(kT) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=0}^{N-1} y(nT)x(nT + kT)$$

When both functions are the same ($x = y$), the cross-correlation becomes the auto-correlation

$$\psi_{xx}(kT) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=0}^{N-1} x(nT)x(nT + kT).$$

The auto-correlation function evaluated at $k = 0$ yields

$$\begin{aligned} \psi_{xx}(0) &= \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=0}^{N-1} x^2(nT) \\ &= \overline{x^2(kT)} \end{aligned}$$

the mean squared value of the signal $x(kT)$.

Since the power spectrum is the Fourier transform of the auto-correlation, we see from Table 2 that

$$\Psi_{xx}(m/NT) = X(m/NT)X(-m/NT)$$

if the signal $x(kT)$ is time limited in the interval $[0, NT]$. Therefore

$$\Psi_{xx}(m/NT) = |X(m/NT)|^2.$$

The Fourier transform of the discrete cross-correlation function is

$$\Psi_{xy}(m/NT) = X(-m/NT)Y(m/NT)$$

and is sometimes called cross power spectrum or cross-periodogram.

Lastly, consider the discrete system of Figure 13b with a random input whose power spectrum is known. We may find the mean squared value of the filter's output by the following:

$$\overline{y^2(nT)} = \frac{1}{2\pi j} \int_{\Gamma} \Psi_{xx}(z) D(z) D(1/z) dz/z$$

where

Γ = the unit circle

$$\Psi_{xx}(z) = \sum_{k=0}^{\infty} \psi_{xx}(kT) z^{-k}$$

= power spectrum

REFERENCES

- [1] B. C. Kuo, Analysis and Synthesis of Sampled-Data Control Systems, Englewood Cliffs, New Jersey, Prentice Hall, Inc. 1963.
- [2] R. M. Golden, "Course Notes -- Designing Digital Filters z-Transforms and Fourier Analysis," National Electronics Conference, St. Charles, Ill., June, 1969.
- [3] A. P. Sage and S. L. Smith, "Real-Time Digital Simulation for Systems Control," Proc. of IEEE, Vol. 54, No. 12, Dec., 1966, pp. 1802-1812.
- [4] H. T. Nagle, Jr., "The Organization of a Special-Purpose Computer to Implement a Generalized Digital Filter for Sampled-Data Control Systems," Doctoral Dissertation, Auburn University, Auburn, Ala., June 3, 1968.
- [5] P. M. DeRusso, R. J. Roy and C. M. Close, State Variables for Engineers, New York, N. Y., John Wiley and Sons, Inc., 1965.
- [6] G. K. McAuliffe, "Course Notes - The Fast Fourier Transform and Some of Its Applications," National Electronics Conference, St. Charles, Ill., June, 1969.
- [7] A. R. M. Naton, Introduction to Variational Methods in Control Engineering, New York, Pergamon Press, 1965.
- [8] G. Williamson, "Optimal Controllers for Homing Missiles," Report #RE-TR-68-15, U.S. Army Missile Command, Redstone Arsenal, Alabama, Sept., 1968.
- [9] R. E. Kalman and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," J. Basic Eng., March, 1961, pp. 95-108.
- [10] T. C. Hsia, "On Synthesis of Optimal Digital Filters," Proc. First Asilomar Conference on Circuits & Systems, Nov., 1967.
- [11] J. B. Slaughter, "Quantization Errors in Digital Control Systems," IEEE Transactions on Automatic Control, Vol. AC-9, January, 1964, pp. 70-74.
- [12] H. T. Nagle, Jr., "Comments on 'A Least Upper Bound in Quantization Error,'" IEEEETAC, Vol. AC-14, No. 4, Aug., 1969.
- [13] B. Widrow, "Statistical Analysis of Amplitude Quantized Sampled-Data Systems," AIEE Transactions on Applications and Industry, No. 52, January, 1961.

- [14] C. A. Halijak, "The (w,v) -Transform," Proceedings of the IEEE 1972 Region 3 Convention, Knoxville, Tennessee, April 10-12, 1972, pp. C4.1-C4.3.
- [15] R. E. Scott, "An Improved Phase Lock Loop Derived from Ideal Single Sideband Modulation," Ph.D. Dissertation, Univ. of Denver, Denver, Colorado, June, 1966, pp. 20-27.
- [16] A. J. Monroe, Digital Processes For Sampled Data Systems, John Wiley and Sons, Inc., New York, 1962.

PART TWO

DIGITAL FILTER THEORY

PART TWO: DIGITAL FILTER THEORY

TABLE OF CONTENTS

I.	Digital Filter Categories.	2-1
A.	Nonrecursive Filters	2-1
1.	General.	2-1
2.	Finite Impulse Response Filters.	2-2
3.	Fast Convolution	2-5
4.	Linear Phase Filters	2-7
5.	Frequency Sampling Filters	2-10
6.	Windowing Filters.	2-13
7.	Moving Average Filter.	2-14
8.	Least Mean-Square Digital Filters.	2-17
9.	Least Squares Polynomial Moving Arc Filter	2-19
10.	Digital Inverse Filtering.	2-20
B.	Recursive Digital Filters.	2-23
1.	General.	2-23
2.	Block Recursion.	2-23
3.	Flat Group Delay Digital Filters	2-27
C.	Advanced Topics.	2-30
1.	Complex Digital Filters.	2-30
2.	Randomly Sampled Filters	2-32
3.	Multirate Digital Filtering.	2-36
4.	Two Dimensional Digital Filters.	2-39
5.	Adaptive Digital Filters	2-45
6.	Floating Point Digital Filters	2-45
7.	Optimal Digital Filters.	2-47
8.	Nonlinear Filtering.	2-57
9.	Range Adaptive Digital Filtering	2-57
10.	Random Sample Skipping	2-57
11.	Block-Floating-Point Filters	2-59
12.	Sample-Rate Reduction Digital Filters.	2-60
II.	Transfer Function Synthesis.	2-62
A.	Nonrecursive Filters	2-62
1.	Specification of Frequency Domain Zeroes	2-62
2.	Frequency Sampling	2-64

3.	Windowing.	2-64
4.	Equiripple Filters	2-65
B.	Recursive Filters.	2-66
1.	Direct Synthesis in the Frequency Domain	2-66
2.	Sampled Data Transformations	2-68
3.	Digital Compensators	2-90
4.	Frequency Sampling	2-93
5.	Nonlinear Programming.	2-94
6.	Optimal Digital Equivalent	2-95
C.	Sample Designs	2-101
1.	Bandstop Filter.	2-101
2.	Digital Resonators	2-101
3.	Digital Differentiators.	2-102
4.	Low-Pass Filters	2-103
III.	Coefficient Quantization	2-104
A.	General.	2-104
B.	Instability Thresholds	2-104
C.	Reduced Coefficient Wordlengths.	2-105
IV.	Nonlinearities in Fixed Point Arithmetic	2-108
A.	Quantization Errors.	2-108
1.	Quantizer Types.	2-108
2.	Steady-State Analysis.	2-111
3.	Statistical Analysis	2-116
4.	Quantization Error Bounds.	2-118
5.	Open-Loop vs. Closed-Loop.	2-122
B.	Limit Cycles and Deadband Effects.	2-123
C.	Saturation and Overflow.	2-124
D.	Dynamic Range.	2-125
V.	Nonlinearities in Floating Point Arithmetic.	2-126
A.	Notation	2-126
B.	Error Sources.	2-127

C.	Coefficient Quantization	2-129
D.	Output Error	2-131
VI.	Programming Forms	2-137
A.	Direct Form	2-138
B.	Modified Direct	2-140
C.	Standard Form	2-143
D.	Modified Standard Form	2-145
E.	Canonical Form	2-147
F.	Modified Canonical Form	2-149
G.	Parallel Form	2-152
H.	Cascade Form	2-154
I.	Modified Cascade	2-157
J.	X1 Structure	2-160
K.	X2 Structure	2-162
L.	Summary of Programming Forms	2-164
VII.	Computer Aided Design	2-168
A.	Transfer Function Synthesis	2-168
B.	Coefficient Quantization	2-169
C.	Programming Form Selection	2-169
1.	General	2-169
2.	Flow Charts	2-172
3.	Source Listing	2-174
4.	Summary	2-174
5.	Stored Program Mode	2-176
6.	Special-Purpose Computer Mode	2-178
7.	Closed Loop Comparison	2-181
8.	Conclusion	2-185

VIII. Applications of Digital Filtering.	2-186
REFERENCES.	2-187

I. DIGITAL FILTER CATEGORIES

The generalized transfer function of a digital filter has been shown to be the ratio of two polynomials in z . Generally the coefficients of the polynomials are real numbers which must be determined in some manner to force the digital filter transfer characteristics to meet some criteria. The manner in which the coefficients are found as well as other considerations (for example, implementation details) allow us to categorize digital filters into several classifications. In this chapter we examine non-recursive, recursive, and several other major categories for digital filters.

Non-Recursive Filters

General

Non-recursive digital filters are those whose transfer function can be written as

$$D(z) = \sum_{i=0}^m a_i z^{-i} \quad . \quad (1-1)$$

Non-recursive filters have no feedback terms, and hence they have a finite impulse response. They are sometimes called transversal filters, a name used for delay line filters in radar moving-target-indicator applications.

Much emphasis has been placed on these filters in the literature and several design techniques will now be illustrated.

Finite Impulse Response Filters [1]

Finite impulse response (FIR) filters satisfy equation (1-1).

Consider the first order filter

$$\begin{aligned} H(z) &= \frac{1}{1 - az^{-1}} & |a| < 1 \\ &= 1 + az^{-1} + (az^{-1})^2 + \dots \\ &= \sum_{\ell=0}^{\infty} (az^{-1})^{\ell} . \end{aligned}$$

Suppose we truncate the series to M terms to produce the FIR below

$$H_M(z) = \sum_{\ell=0}^{M-1} (az^{-1})^{\ell} \quad (1-2)$$

Also,

$$H_M(z) = \frac{1 - (az^{-1})^M}{1 - az^{-1}} = \frac{z^M - a^M}{z^{M-1}(z - a)} \quad (1-3)$$

This is another way of expressing the FIR as filter with feedback.

Fig. 1 illustrates the z-plane pole-zero locations for both $H(z)$ and $H_M(z)$ for $M = 8$. Fig. 2a shows the implementation of (1-2); Fig. 2b, (1-3).

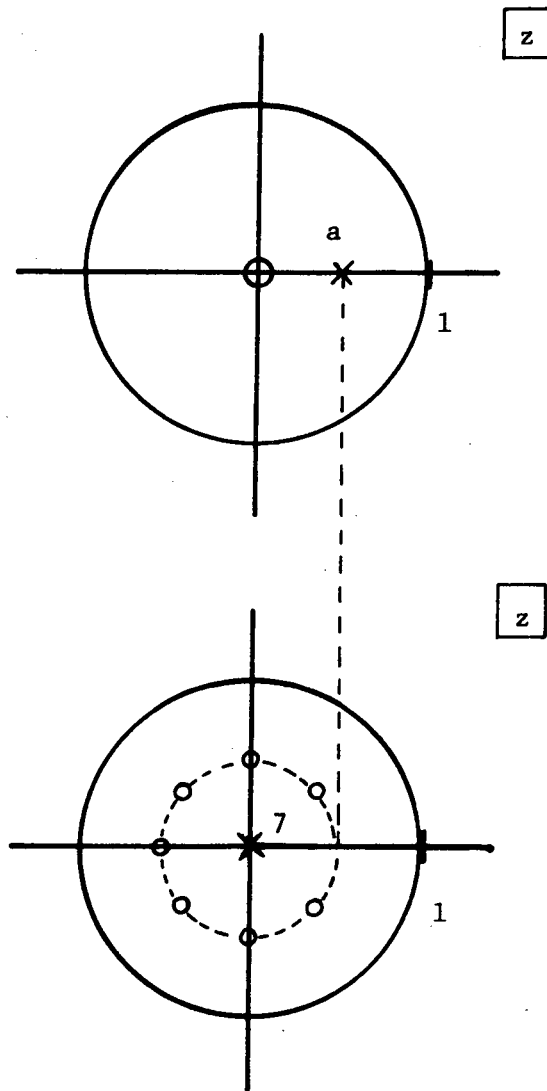
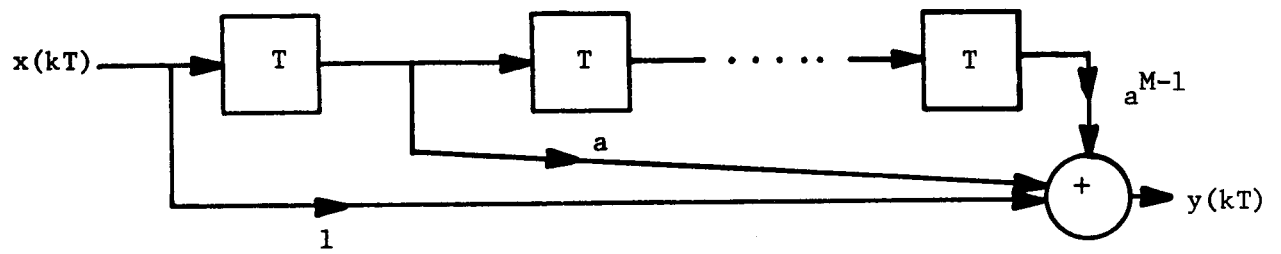
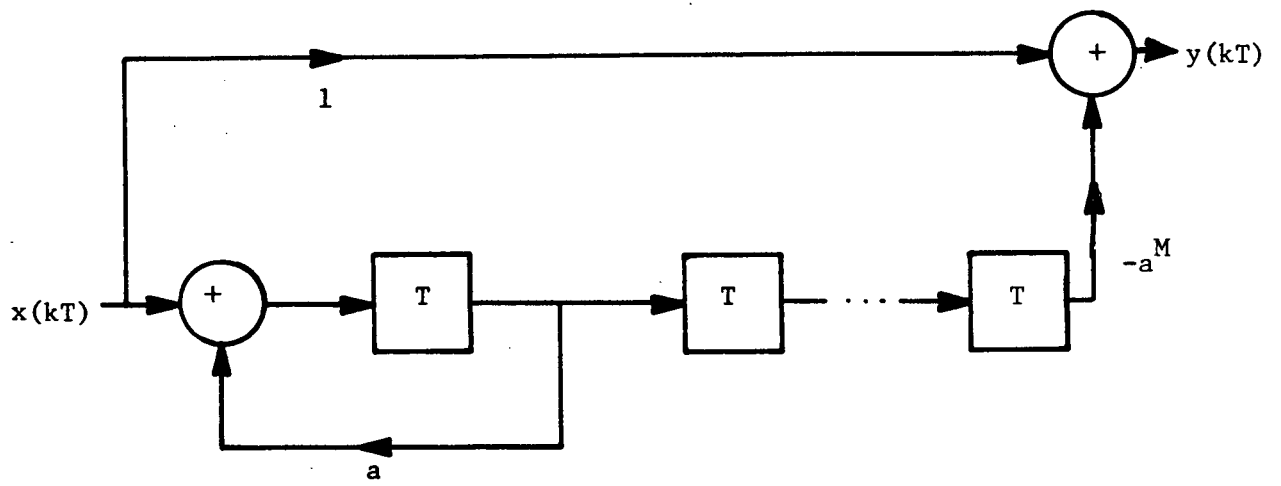


Fig. 1. Pole-zero Plots in the z -plane.



(a)



(b)

Fig. 2. Block Diagrams for (1-2) and (1-3)

Fast Convolution [2]

Fast convolution is a technique which employs the FFT and IFFT to determine the filter output response (see Fig. 3a). Direct convolution is expressed by

$$y(kT) = \sum_{\ell=0}^{N-1} h(\ell T)x(kT - \ell T). \quad (1-4)$$

To calculate N output points, this requires N^2 real multiplications.

For fast convolution

$$y(kT) = \text{IFFT}\left\{H\left(\frac{m}{NT}\right) \cdot \text{FFT}(x(kT))\right\}. \quad (1-5)$$

Here the FFT and IFFT require $2N\log_2 N$ operations each, while the multiplication requires N operations. This totals

$$\# \text{ operations} = N(4 \log_2 N + 1).$$

If each operation (a complex multiplication) is assumed to take approximately 4 real multiplications, the result is

$$\# \text{ multiplications} \approx 16 N \log_2 N. \quad (1-6)$$

Suppose $N = 1024$, then $N^2 \approx 10^6$ and $16 N \log_2 N \approx 1.6 \times 10^5$. Hence, for large numbers of output points, the fast convolution technique is faster than direct convolution.

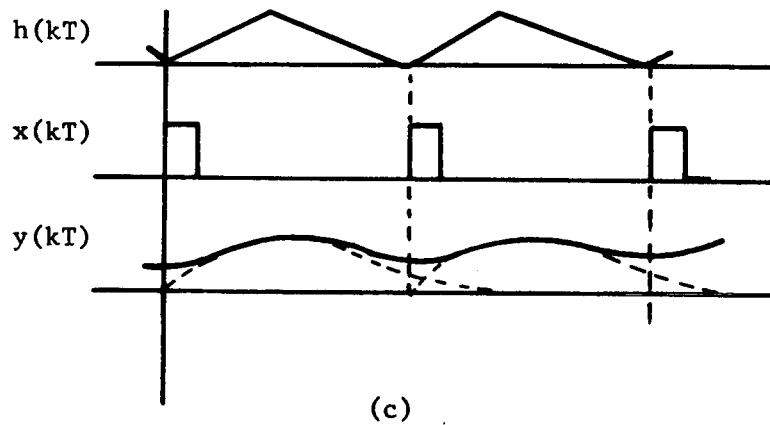
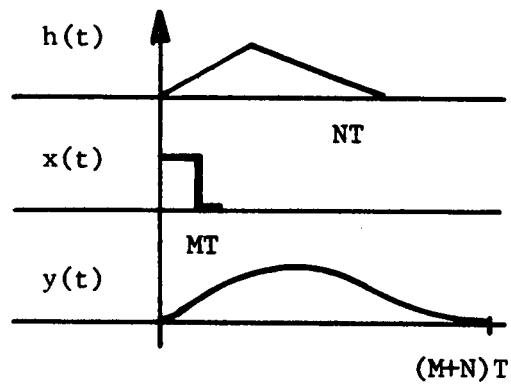
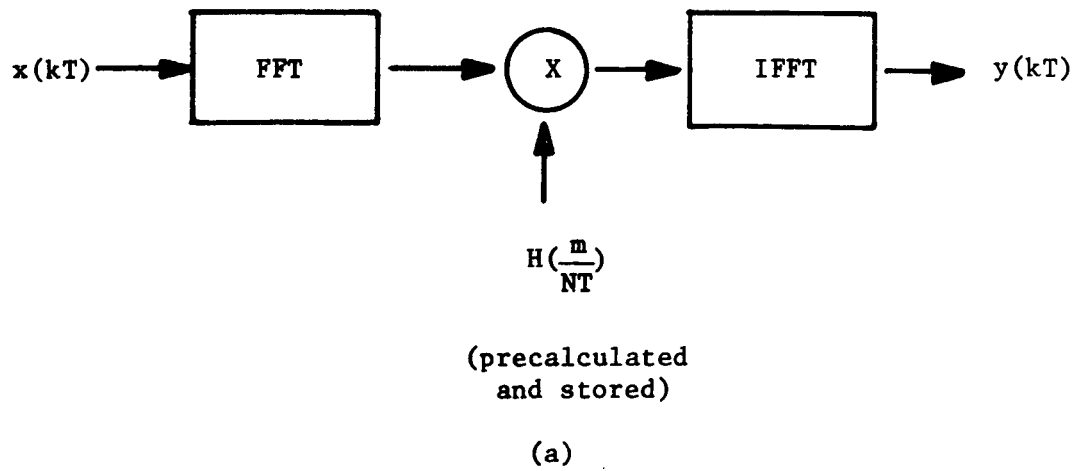


Fig. 3. Fast Convolution

The reader must be careful in using the fast convolution technique because the results can be misleading. Consider the convolution of the analog signals in Fig. 3b. If we sample these signals and use the FFT and IFFT, we are convolving the periodic functions shown in Fig. 3c. Hence the output $y(kT)$ can differ greatly from the desired sequence.

In order to improve the results one may add zeroes into the input and transfer function sample sequences as shown in Fig. 4. Note the improved output response. However, in adding zeroes we have increased the calculation time unless we modify the FFT algorithm.

Linear Phase Filters [3]

A linear phase filter is an FIR filter with exact linear phase. They may be used to approximate an arbitrary magnitude frequency response without causing phase errors. The linear phase filter is good for standard lowpass, bandpass, and highpass filters.

If the number of sample points in a FIR filter is

$$N = 2\tau + 1 ,$$

then linear phase with delay τ is realized if and only if

$$h(kT) = h(NT - T - kT) \quad (1-7)$$

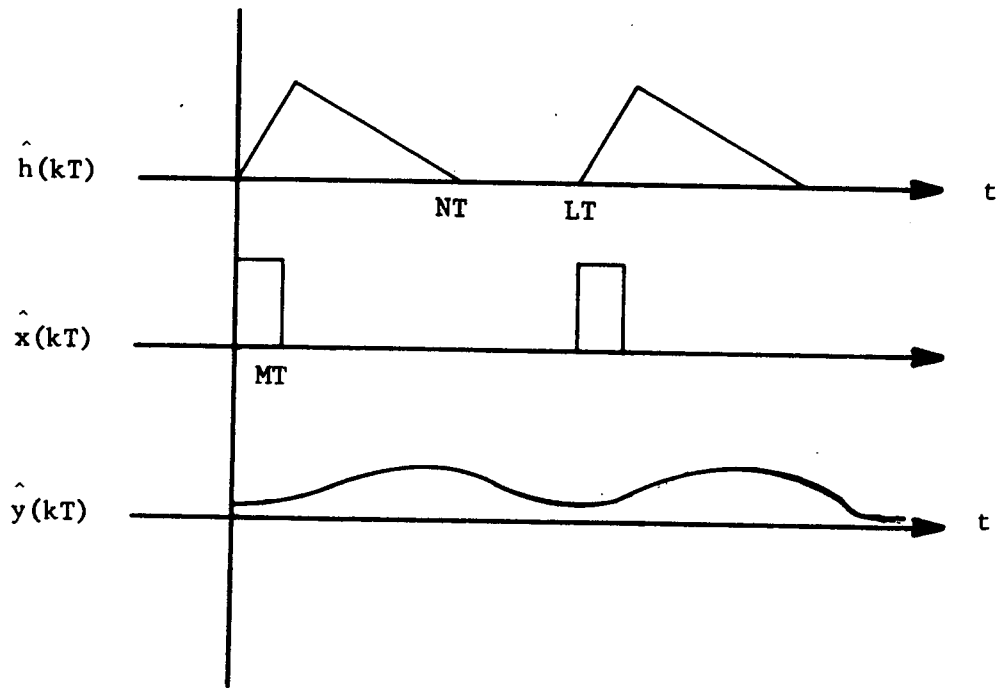


Fig. 4. Add a Zero Sequence

Hence, for N even

1. There is no unique peak in $h(kT)$
2. $h(kT) = h(NT - T - kT)$ $k = 0, (\frac{N}{2}) - 1$
3. The center of symmetry is between $(\frac{N}{2})$ and $(\frac{N}{2}) - 1$.
4. The delay is

$$\tau = \frac{N - 1}{2}$$

the center of symmetry.

For N odd

1. There is a unique peak in $h(kT)$ at $(N - 1)/2$.
2. $h(kT) = h(NT - T - kT)$ $k = 0, (N - 1)/2$
3. The center of symmetry is at $(N - 1)/2$
4. The delay is

$$\tau = \frac{N - 1}{2} ,$$

the center of symmetry.

If the above conditions are met, the frequency samples $H(\frac{m}{NT})$ will be given by

$$H(\frac{m}{NT}) = |H(\frac{m}{NT})| e^{j\theta_m}$$

where, for N even

$$\theta_m = -\frac{2\pi}{N} m\tau \quad m = 0, (\frac{N}{2}) - 1$$

(1-8)

$$\theta_m = \frac{2\pi}{N} (N - m)\tau \quad m = (\frac{N}{2}), N$$

and for N odd

$$\theta_m = -\frac{2\pi}{N} m\tau \quad m = 0, (N-1)/2 \quad (1-9)$$

$$\theta_m = \frac{2\pi}{N} (N-m)\tau \quad m = (\frac{N}{2}) + 1, N.$$

and

$$H(\frac{1}{2T}) = H(\frac{f_s}{2}) = 0. \quad (1-10)$$

This concludes our brief description of linear phase filters.

Frequency Sampling Filters [3]

The term frequency sampling filters refers to a class of digital filters specified by sample points in the frequency domain and implemented in the manner of Fig. 5. Many techniques have been suggested for choosing the sample points

$$H(\frac{m}{NT}) = |H_m| e^{j\theta_m} \quad m = 0, N-1 \quad (1-11)$$

including optimization techniques which adjust the points in the transition region to give a good ripple between sample points. For real impulse response filters

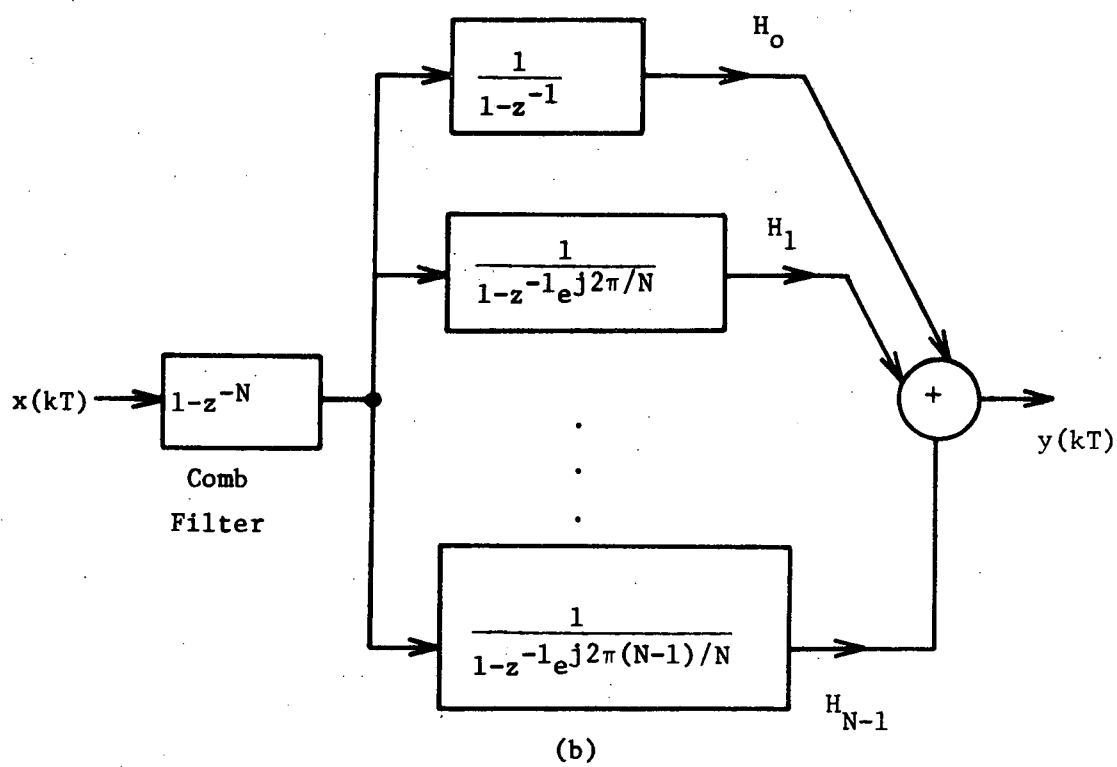
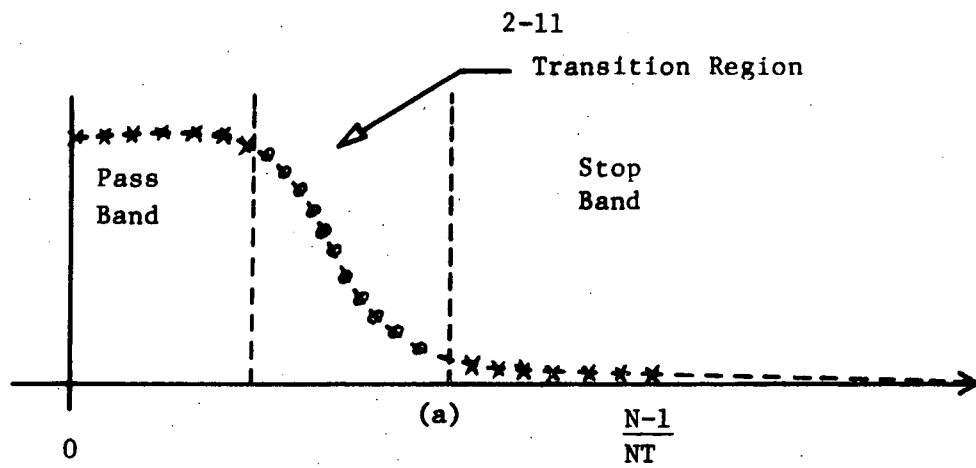


Fig. 5. Frequency Sampling Filter

$$|H_m| = |H_{N-m}|$$

$$\theta_m = -\theta_{N-m}.$$

(1-12)

By the IDFT

$$h(kT) = \sum_{m=0}^{N-1} H\left(\frac{m}{NT}\right) e^{j(2\pi/N)mk} \quad k = 0, N-1$$

and

$$H(z) = \sum_{k=0}^{N-1} h(kT) z^{-k}$$

where

$$\left. H(z) \right|_{z = e^{j2\pi m/N}} = H\left(\frac{m}{NT}\right).$$

Then

$$H(z) = \sum_{k=0}^{N-1} \left[\sum_{m=0}^{N-1} H\left(\frac{m}{NT}\right) W^{mk} \right] z^{-k}$$

or

$$H(z) = (1 - z^{-N}) \sum_{m=0}^{N-1} \frac{H\left(\frac{m}{NT}\right)}{1 - z^{-1} W^m} \quad (1-13)$$

where

$$W = e^{j2\pi/N}.$$

Equation (1-13) is the motivation behind the frequency sampling implementation illustrated in Fig. 5b.

Windowing Filters [4]

In equation (1-2) a truncation was performed (a fairly drastic measure) to produce a FIR filter from an infinite impulse response function. Windowing is the process of orderly termination of an infinite series by truncating the series and adjusting the remaining terms to mask the truncation effects. The transfer function for the FIR is given in equation (1-1); its output response is

$$y(kT) = \sum_{i=0}^m a_i x(kT - iT).$$

Briefly stated, the problem is to find the coefficients a_i of the FIR filter $H(z)$ such that

$$H(e^{j\omega T}) \approx F(e^{j\omega T})$$

where $F(e^{j\omega T})$ is some specified desired frequency response. The design procedure is outlined below:

1. From $F(e^{j\omega T})$, use the IFFT algorithm to find $f(kT)$.
2. Multiply $f(kT)$ by a window function $w(kT)$, or

$$h(kT) = f(kT)w(kT) \quad (1-15)$$

The process is outlined in Fig. 6. Multiplication in the time domain is convolution in the frequency domain, and hence

$$H(f) = F(f) * W(f) .$$

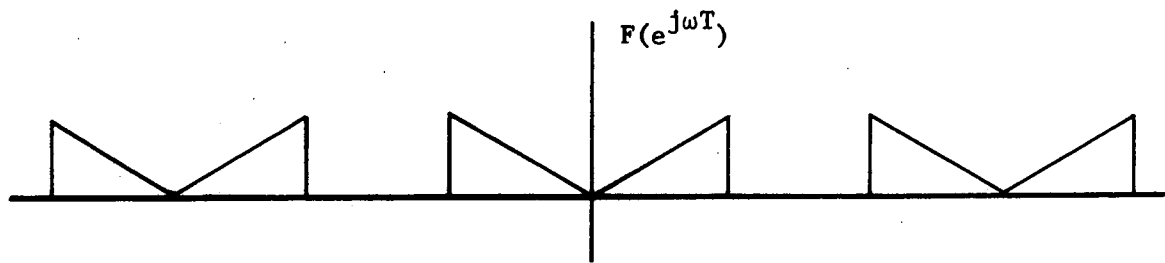
The window function shown is a rectangular one which duplicates the truncation process. Notice the ringing effect in Fig. 6c. The sidelobes for this window are about 20%. Fig. 7 illustrates two other windows. The triangular one reduced the sidelobes to about 4%. The raised cosine window is the best one shown. Its function is

$$w(t) = \alpha + (1 - \alpha) \cos \frac{\pi t}{A} \quad (1-16)$$

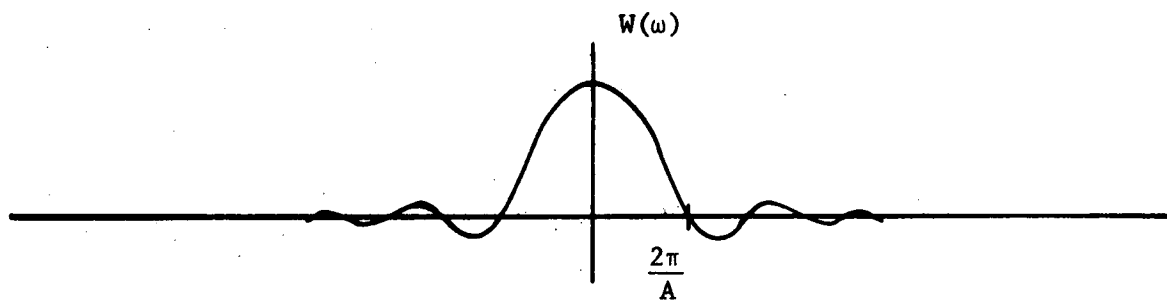
If $\alpha = 0.50$ it is called a Hamming window. The optimal value of α is about 0.54. This value yields the Hanning window and reduces the sidelobes to about 1%.

Moving Average Filter [5]

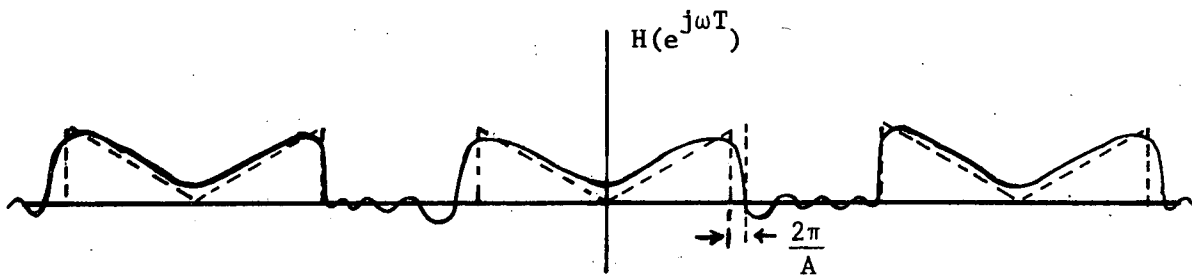
A moving average filter is a FIR filter which calculates the average of the N most recent observations of the input:



(a) Desired Frequency Response



(b) Window Function



(c) Windowing Filter

Fig. 6. Windowing Filter Construction

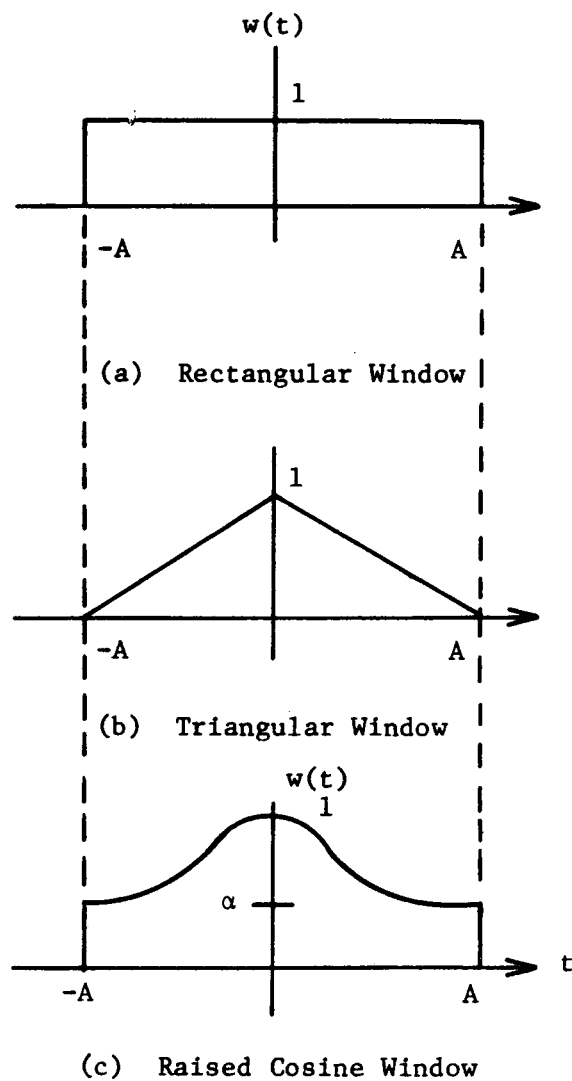


Fig. 7. Window Functions

$$y(kT) = \frac{1}{N} \sum_{\ell=0}^{N-1} x(kT - \ell T)$$

and

$$H(z) = \frac{1}{N} \sum_{\ell=0}^{N-1} z^{-\ell}$$

In another form

$$H(z) = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}} \quad (1-17)$$

Least Mean-Square Digital Filters [6,7]

Assume that the filter input is $x(nT)$, a random signal whose autocorrelation $R_{xx}(t)$ is known, and that the crosscorrelation $R_{dx}(t)$ of this input and a desired output $d(nT)$ is also specified. Let the impulse response of the filter be $g(kT)$, and its output, $z(kT)$. Allowing a shifted time scale,

$$z(nT) = \sum_{k=-M}^N x(nT - kT)g(kT).$$

Define the signal D to be expected value of the difference in the actual and desired filter outputs squared:

$$D = E[d(nT) - z(nT)]^2.$$

By definition

$$R_{xx}(t) = E \{x(t + \tau)x(\tau)\}$$

$$R_{dx}(t) = E \{d(t + \tau)x(\tau)\} \quad (1-18)$$

$$R_{dd}(t) = E \{d(t + \tau)d(\tau)\}$$

Substitution of these relations into D yields

$$\begin{aligned} D = R_{dd}(0) - 2 \sum_{n=-M}^N R_{dx}(nT)y(nT) \\ + \sum_{k=-M}^N \sum_{n=-M}^N R_{xx}(kT - nT)g(kT)g(nT). \end{aligned} \quad (1-19)$$

The purpose of the least-mean squares filter is to minimize D by choosing $g(nT)$. Hence, if one takes the partial derivative of D with respect to $g(nT)$ and sets the result to zero, the following solution is generated

$$R_{dx}(nT) = \sum_{k=-M}^N R_{xx}(kT - nT)g(kT) \quad -M \leq n \leq N \quad (1-20)$$

In equation (1-20), all quantities are known except $g(kT)$. Hence, the filter weights may be calculated from (1-20). Least mean-squares filters are sometimes called digital Wiener filters.

Least Squares Polynomial Moving Arc Filter [5]

The problem here is to solve for the coefficients a_i of a polynomial to best fit the input data $y(t_i)$ in a least squares sense. Each input point is approximated by

$$\begin{aligned} y(t_i) &= a_0 + a_1 t_i + a_2 t_i^2 + \cdots + a_d t_i^d \\ &= \sum_{k=0}^d a_k t_i^k \end{aligned}$$

If the input samples are evenly spaced, $t_i = iT$ and

$$y(iT) = \sum_{k=0}^d a_k (iT)^k$$

For n input samples define

$$S = \sum_{i=0}^n \left[\sum_{k=0}^d a_k (iT)^k - y(iT) \right]^2$$

In order to minimize S by choosing a_k , one may take the partial derivative

$$\frac{\partial S}{\partial a_k} = \sum \left\{ 2 \left[\sum_{k=0}^d a_k (iT)^k - y(iT) \right] \sum_{\ell=0}^d (\ell T)^\ell \right\} = 0$$

This expression reduces to

$$\sum_{k=0}^d \sum_{\ell=0}^d a_k (iT)^k (\ell T)^\ell = \sum_{\ell=0}^d y(iT) (\ell T)^\ell$$

Written in matrix form

$$CA = B$$

or

$$A = C^{-1}B. \tag{1-21}$$

In (1-21) the matrix C^{-1} represents the filter itself (whose coefficients are precomputed) and B represents the system input. The output is A which represents the polynomial coefficients a_k .

Another form of polynomial filtering termed exponential filtering allows the polynomial to grow by one term as each new input occurs. Such schemes are called "growing memory" filters.

Digital Inverse Filtering [8,9]

Digital inverse filtering is a special case of least mean-square filtering as described in equation (1-20). Suppose that the desired filter output is

$$d(kT) = 1, 0, 0, 0, \dots$$

the discrete impulse function. Hence the crosscorrelation

$$R_{dx}(nT) = x(0), 0, 0, 0, \dots$$

which can be scaled to unity ($x(0) = 1$). Equation (1-20) with $M = 0$ then becomes

$$\begin{bmatrix} r_0 & r_1 & r_2 & \dots & r_N \\ r_1 & r_0 & r_1 & & r_{N-1} \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \\ r_N & r_{N-1} & r_{N-2} & & r_0 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ \cdot \\ \cdot \\ \cdot \\ g_N \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad (1-22)$$

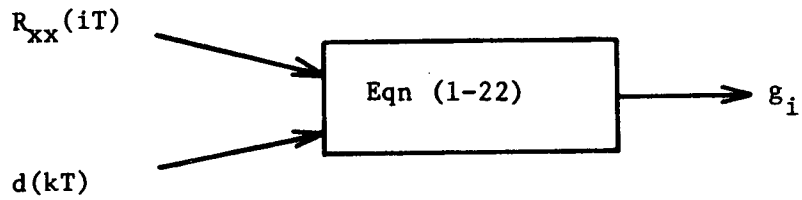
where

$$r_i = r_{-i} = R_{xx}(iT) \quad .$$

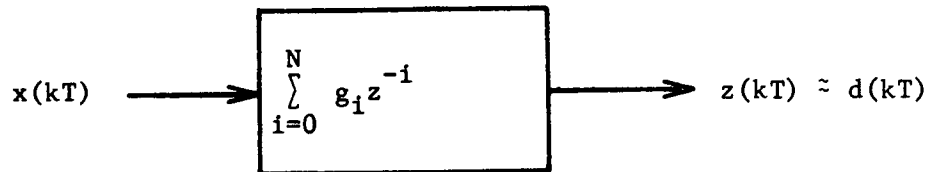
If the filter coefficients g_i are used in an FIR filter

$$H(z) = \sum_{i=0}^N g_i z^{-i}$$

and the random signal $x(nT)$ is applied to the input, the output will be a digital impulse function. Therefore, $H(z)$ is said to be an inverse digital filter. The calculations involved are shown in Fig. 8.



(a) Filter Design



(b) Filter Application

Fig. 8. Digital Inverse Filtering

Recursive Digital Filters

General

A recursive digital filter is a filter with feedback which, in general, has an infinite impulse response. Its transfer function is

$$H(z) = \frac{\sum_{i=0}^n a_i z^{-i}}{1 + \sum_{i=1}^n b_i z^{-i}} \quad (1-23)$$

where at least one a_i and b_i is not zero.

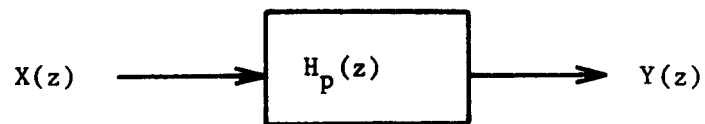
Recursive filters generally require fewer terms (lower order) than a non-recursive filter with similar characteristics. Higher order recursive filters are usually factored into second order stages which are either cascaded or paralleled.

Block Recursion [10]

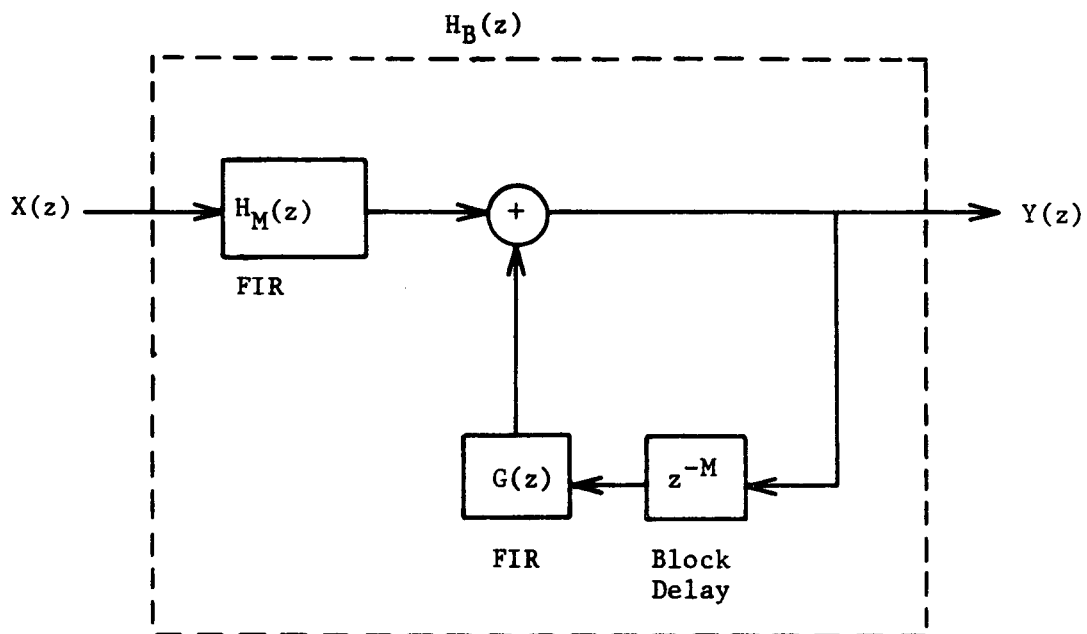
One technique for implementing a desired recursive digital filter of the form

$$H_p(z) = \frac{1}{D(z)} \quad (1-24)$$

is called block recursion and is shown in Fig. 9. The implementation in Fig. 9b is



(a) The Desired Filter



(b) The Implemented Filter

Fig. 9. Block Recursion

$$H_B(z) = \frac{H_M(z)}{1 - z^{-M}G(z)} \quad (1-25)$$

But the desired filter is

$$H_p(z) = \frac{1}{\prod_{i=1}^m (1 - z_i z^{-1})} = \sum_{i=1}^m \frac{a_i}{1 - z_i z^{-1}},$$

where z_i are the poles of the function $H_p(z)$.

The finite impulse response filter $H_M(z)$ is found by truncating each component of $H_p(z)$ to M terms, or

$$\begin{aligned} H_M(z) &= \sum_{i=1}^m \frac{a_i [1 - (z_i z^{-1})^M]}{1 - z_i z^{-1}} \\ &= H_p(z) - z^{-M} \sum_{i=1}^m \frac{a_i z_i^M}{1 - z_i z^{-1}} \\ &= \frac{1 - z^{-M}Q(z)}{D(z)} \end{aligned} \quad (1-26)$$

where

$$Q(z) = \sum_{i=1}^m \frac{a_i z_i^M D(z)}{1 - z_i z^{-1}}.$$

Thus,

$$H_M(z)D(z) = 1 - z^{-M}Q(z)$$

and

$$Q(z) = z^M(1 - H_M(z)D(z)) \quad (1-27)$$

where $Q(z)$ is a polynomial of order $M-1$. From the above relations it is clear that if $G(z)$ in $H_B(z)$ is chosen as $Q(z)$, then

$$\left. \begin{array}{l} H_B(z) \\ G(z) = Q(z) \end{array} \right| = H_p(z) \quad (1-28)$$

and the block recursive implementation exactly produces $H_p(z)$, the desired filter. Thus, we have shown that a recursive filter can be implemented using one FIR $H_M(z)$ in the feed forward path and one FIR $G(z)$ in the feedback path, where

$$\begin{aligned} H_M(z) &= \text{truncated version of } H_p(z) \\ G(z) &= z^M(1 - H_M(z)D(z)). \end{aligned} \quad (1-29)$$

Some researchers have used the FFT to implement the two FIR filters [11-13].

Example. Consider the filter

$$H_p(z) = \frac{1}{1 + az^{-1} + bz^{-2}} = \frac{1}{D(z)}$$

and let $M = 3$

$$1 + az^{-1} + bz^{-2} \sqrt{\frac{1 - az^{-1} + (a^2 - b)z^{-2}}{1 + az^{-1} + bz^{-2} - az^{-1} - bz^{-2} - az^{-1} - a^2z^{-2} - abz^{-3}}} \\ \frac{(a^2 - b)z^{-2} + abz^{-3}}$$

Hence

$$H_3(z) = 1 - az^{-1} + (a^2 - b)z^{-2}$$

$$G(z) = z^3(1 - H_3(z)D(z))$$

$$= (2ab - a^3) + (b^2 - a^2b)z^{-1}.$$

One can check the impulse response of $H_p(z)$ by dividing the denominator into the numerator and comparing it with $H_p(z)$.

Flat Group Delay Digital Filters [14]

In order to achieve a linear-phase digital filter one must choose a non-recursive structure. However, when the order of the non-recursive filter is unacceptably larger, one is led to approximate the linear phase using a recursive filter design whose error norm is the maximally flat criteria.

Consider the recursive filter

$$H(z) = \frac{1 + \sum_{i=1}^n a_i}{1 + \sum_{i=1}^n a_i z^{-i}} \quad (1-30)$$

whose d.c. gain is unity. The phase response ($T = 1$) is given by

$$\tan^{-1} \left[\frac{\sum_{i=0}^n a_i \sin i \omega}{\sum_{i=0}^n a_i \cos i \omega} \right] = \phi(\omega) \quad (1-31)$$

The ideal phase ($-\omega\tau$), where τ is the desired delay is approximated by minimizing

$$\delta(\omega) = -\omega\tau - \phi(\omega) \quad (1-32)$$

or

$$\epsilon(\omega) = \tan(\delta(\omega)) = -\tan \omega\tau - \tan (\phi(\omega)) .$$

The procedure is to make $\epsilon(\omega)$ vanish at d.c., together with its derivatives up to some order depending on n .

The solution yields the filter

$$H(z) = \frac{2n!}{n!} \frac{1}{2n \prod_{i=n+1}^{\infty} (2\tau + i)}$$

(1-33)

$$\cdot \frac{1}{\sum_{k=0}^n (-1)^k \binom{n}{k} \prod_{i=0}^n \frac{2\tau + i}{2\tau + k + i} z^{-k}}$$

which is stable for all finite positive values of τ .

Advanced Topics

In addition to the simple division of digital filters into recursive or non-recursive categories, there are many other ways of identifying their characteristics.

Complex Digital Filters [15]

A complex digital filter has a complex input $x(nT)$, a complex output $y(nT)$, and a complex transfer function $H(z)$. An example is shown in Fig. 10. A lowpass envelope is centered at f_c by replacing z by $e^{-j\omega_c T} z = \gamma z$ in $H(z)$.

$$H(z) = \frac{\sum_{\ell=0}^n a_{\ell} z^{-\ell}}{1 + \sum_{\ell=1}^n b_{\ell} z^{-\ell}}$$

Hence

$$H_S(z) = \frac{\sum_{\ell=0}^n \gamma^{\ell} a_{\ell} z^{-\ell}}{1 + \sum_{\ell=1}^n \gamma^{\ell} b_{\ell} z^{-\ell}} \quad . \quad (1-34)$$

Complex digital filters have application in communication and information theory, signal detection, randomly time-invariant channels, etc. In one application they are used to generate the Hilbert transform of a real signal $x(nT)$.

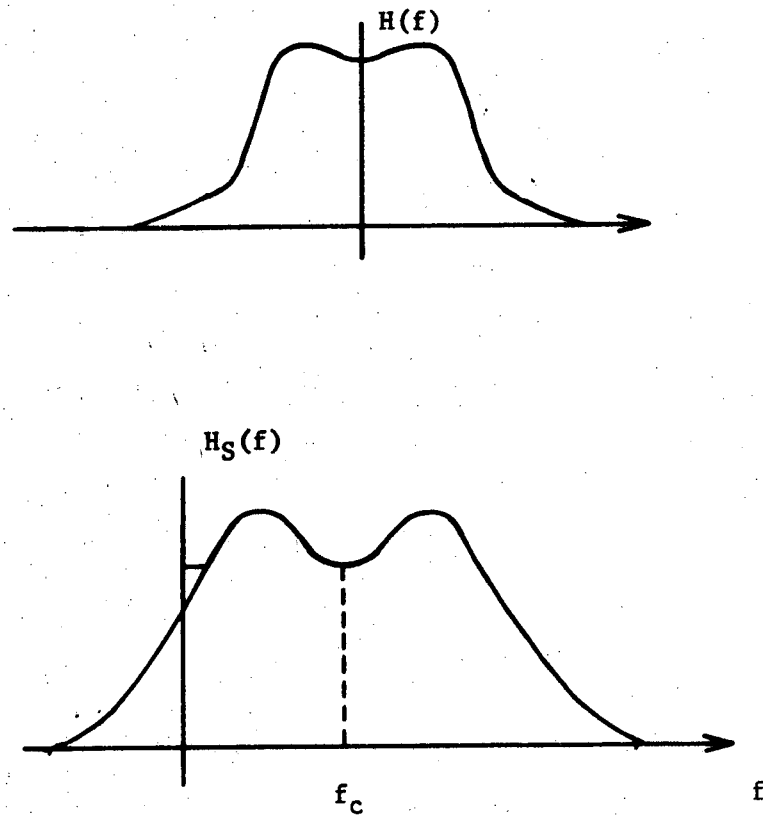


Fig. 10. Complex Bandpass Filter

Randomly Sampled Filters [16]

A randomly sampled digital filter $H(z)$ takes input samples of the analog input signal $x(t)$ at some random time

$$nT \leq t_n \leq nT + T$$

and stores them in an input buffer. The numbers $x(t_n)$ are then fed to the filter hardware as evenly spaced samples $x(nT)$. Hence, the direct convolution of $x(nT)$ and $h(nT)$ produces the output $y(nT)$ which is interpreted as $y(t_n)$. The question arises what errors are generated by the random sampling?

Define

$$t_n = (n + Z_n)T$$

$$-\Delta \leq Z_n \leq \Delta$$

(1-35)

$$\Delta = \frac{\alpha T}{2}$$

$$0 \leq \alpha < 1$$

where Z_n is a random variable. Also define

$$x(t_n) = \hat{x}_n = x(n + Z_n)$$

(1-36)

$$x(nT) = x_n.$$

If we expand \hat{x}_n in a Taylor series about z_n

$$\hat{x}_n = x_n + \dot{x}_n z_n + 1/2 \ddot{x}_n z_n^2 + \dots$$

where $\dot{x}_n = \frac{d}{dt} x_n$, and define an input error ξ_n

$$\xi_n = \hat{x}_n - x_n = \dot{x}_n z_n + 1/2 \ddot{x}_n z_n^2 + \dots$$

The output error due to random sampling is defined as

$$e_n = \hat{y}_n - y_n = \sum_{i=0}^n h_{n-i} \xi_i$$

for a non-recursive filter. Other pertinent relations are

$$v^2 = E(z_n^2)$$

$$\eta^4 = E(z_n^4)$$

$$E(\xi_n) = 1/2 \ddot{x}_n v^2 + \dots$$

(1-37)

$$E(\xi_n^2) = \dot{x}_n^2 v^2 + (1/4 \ddot{x}_n^2 + 1/3 \dot{x}_n \ddot{x}_n) \eta^4 + \dots$$

$$E(e_n) = \sum_{i=0}^n h_{n-i} E(\xi_i)$$

$$E(e_n^2) = E^2(e_n) + \sum_{i=0}^n h_{n-i}^2 \text{var}(\xi_i) .$$

The frequency response error is determined for sinusoid inputs

$$x(t) = \cos \omega t \quad 0 \leq \omega \leq N_f$$

where N_f is the Nyquist frequency π/T . The expected values of the output steady state errors are

$$\begin{aligned} E(e_n)_{ss} &= (-1/2\omega^2 v^2) H(e^{j\omega}) \cos n\omega \\ E(e_n^2)_{ss} &= E^2(e_n)_{ss} + \left(\sum_{i=0}^n h_i^2 \right) \left(\frac{\omega^2 v^2}{2} - \frac{\omega^4 v^4}{24} - \frac{\omega^4 v^4}{8} \right) \\ &\quad - \left(\sum_{i=0}^n h_i^2 e^{-ij2\omega} \right) \left(\frac{\omega^2 v^2}{2} - \frac{7\omega^4 v^4}{24} - \frac{\omega^4 v^2}{8} \right) \cos 2n\omega \end{aligned}$$

The physical interpretation of the results is shown in Fig. 11, where

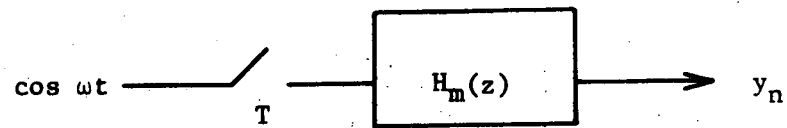
$$E(H_m(e^{j\omega})) = (1 - 1/2\omega^2 v^2) H(e^{j\omega}) \quad (1-38)$$

In random sampling only the expected amplitude response is distorted while the expected phase response is unchanged. The noise to signal ratio for noise generated by random sampling is approximated by

$$NSR = 10 \log_{10} \left(\frac{\sum_{i=0}^n h_i^2 \omega^2 v^2}{|H(e^{j\omega})|^2} \right) \quad (1-39)$$



(a) Exact Model



(b) Approximate Model

Fig. 11. Randomly Sampled Filter

Random sampling finds wide application in time-sharing filters, radar filters, and faulty samplers - all samples are faulty to some extent.

Example. [9]

$$H(z) = \frac{0.1}{1 - .9z^{-1}}$$

and Z_n has a rectangular distribution with $\alpha = 0.1$, or 10% jitter in the input sampler. The curves of Fig. 12 illustrate that as frequency increases, the noise component increases making the filter unusable above $\omega/N_f = 0.3$

Multirate Digital Filtering [17]

A multirate digital filter is one in which the samplers for the input and output are operating at different rates, one usually being an integral multiple of the other. Much analysis of multirate sampled data control systems has been treated in the open literature. Here we examine three configurations of multirate filters demonstrated in Fig. 13. Solutions for the sampled output frequency responses are

$$W^{**}(j\omega) = \frac{1}{KT} \sum_{n=-\infty}^{\infty} G(j\omega + j \frac{2\pi n}{KT}) R^*(j\omega + j \frac{2\pi n}{KT})$$

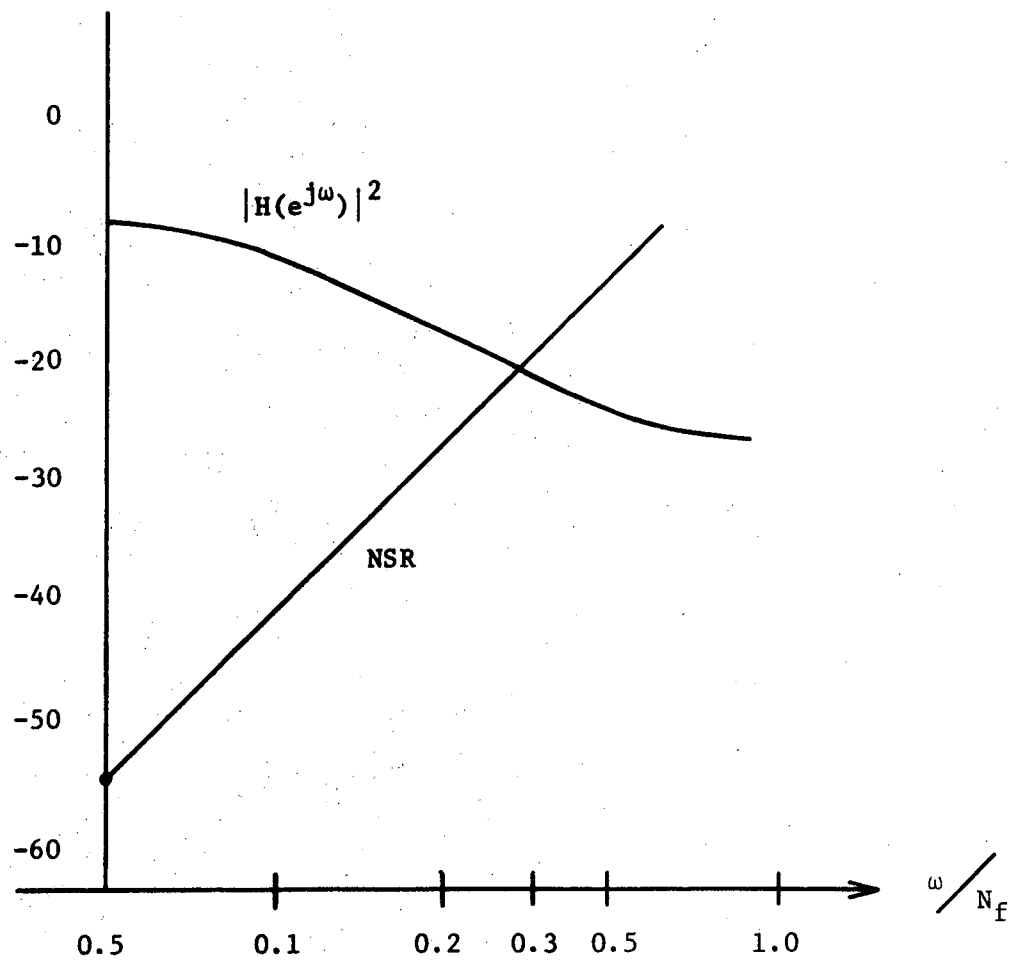
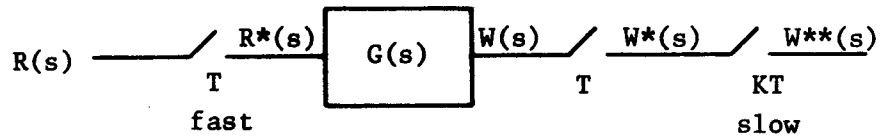
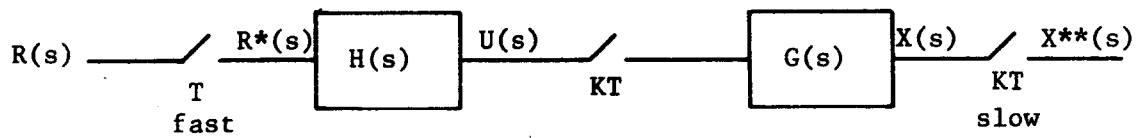


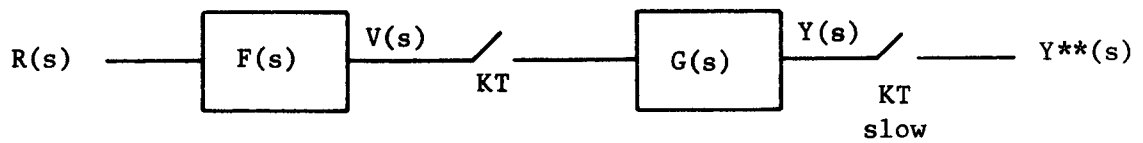
Fig. 12. Random Sampling Example



(a) Typical Multirate Filter



(b) Digital Prefilter



(c) Analog Prefilter

Fig. 13. Multirate Digital Filters

$$X^{**}(j\omega) = \frac{1}{KT} \sum_{n=-\infty}^{\infty} \left[G^{**}(j\omega) H(j\omega + \frac{j2\pi n}{KT}) \right] R^{*}(j\omega + \frac{j2\pi n}{KT}) \quad (1-40)$$

$$Y^{**}(j\omega) = \frac{1}{KT} \sum_{n=-\infty}^{\infty} \left[G^{**}(j\omega) F(j\omega + \frac{j2\pi n}{KT}) \right] R(j\omega + \frac{j2\pi n}{KT})$$

These expressions simplify greatly if the filter function $G(s)$ is band limited

$$|G(j\omega)| = 0 \quad |\omega| \geq \frac{2\pi}{KT} \quad (1-41)$$

The functions $H(s)$ and $G(s)$ represent prefilters used to band limit the input signal $r(t)$ to prevent frequency aliasing.

Two Dimensional Digital Filters [18,19]

Two dimensional digital filters are used in digital image processing. They are used to transform characteristics in photographs or CRT images. The transformation is described by

$$H(z_1, z_2) = \frac{\sum_{m=0}^p \sum_{n=0}^q a_{mn} z_1^m z_2^n}{\sum_{m=0}^p \sum_{n=0}^q b_{mn} z_1^m z_2^n} = \frac{A(z_1, z_2)}{B(z_1, z_2)} \quad (1-42)$$

and

$$z_1 = e^{-s_1 A}$$

$$z_2 = e^{-s_2 B}$$

where s_1 and s_2 are Laplace variables; A and B are the sampling intervals in the x and y planar coordinates of the image being processed. The two dimensional filter may also be expressed as

$$H(z_1, z_2) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} h_{mn} z_1^m z_2^n \quad (1-43)$$

where h_{mn} is the impulse response.

Let us consider the stability of a two dimensional digital filter.

For a stable filter

$$\sum_{m=0}^{\infty} \sum_{n=0}^{\infty} |h_{mn}| < \infty. \quad (1-44)$$

A two dimensional digital filter is stable if and only if no value of z_1 and z_2 exist such that

$$B(z_1, z_2) = 0, \text{ and}$$

$$|z_1| \leq 1, \text{ and}$$

$$|z_2| \leq 1.$$

Equivalent conditions are listed below: $H(z_1, z_2)$ is stable if and only if

- 1) The map $B(z_1, z_2) = 0$ of the unit circle $|z_1| = 1$ to the z_2 plane is outside the unit circle $|z_2| = 1$.
- 2) No point in $|z_1| \leq 1$ maps into $z_2 = 0$; or $z_2 = 0$ maps outside the unit circle in the z_1 plane.

Example. Given the two-dimensional filter

$$H(z_1, z_2) = \frac{1}{1 + az_1 + bz_2}$$

we may set the denominator to zero.

$$B(z_1, z_2) = 1 + az_1 + bz_2 = 0$$

to determine the following map

$$z_2 = -\frac{1}{b} - \frac{a}{b} z_1$$

Condition 1 is shown in Fig. 14.

$$r = \left| \frac{a}{b} \right|$$

$$-\frac{1}{b} + r < -1$$

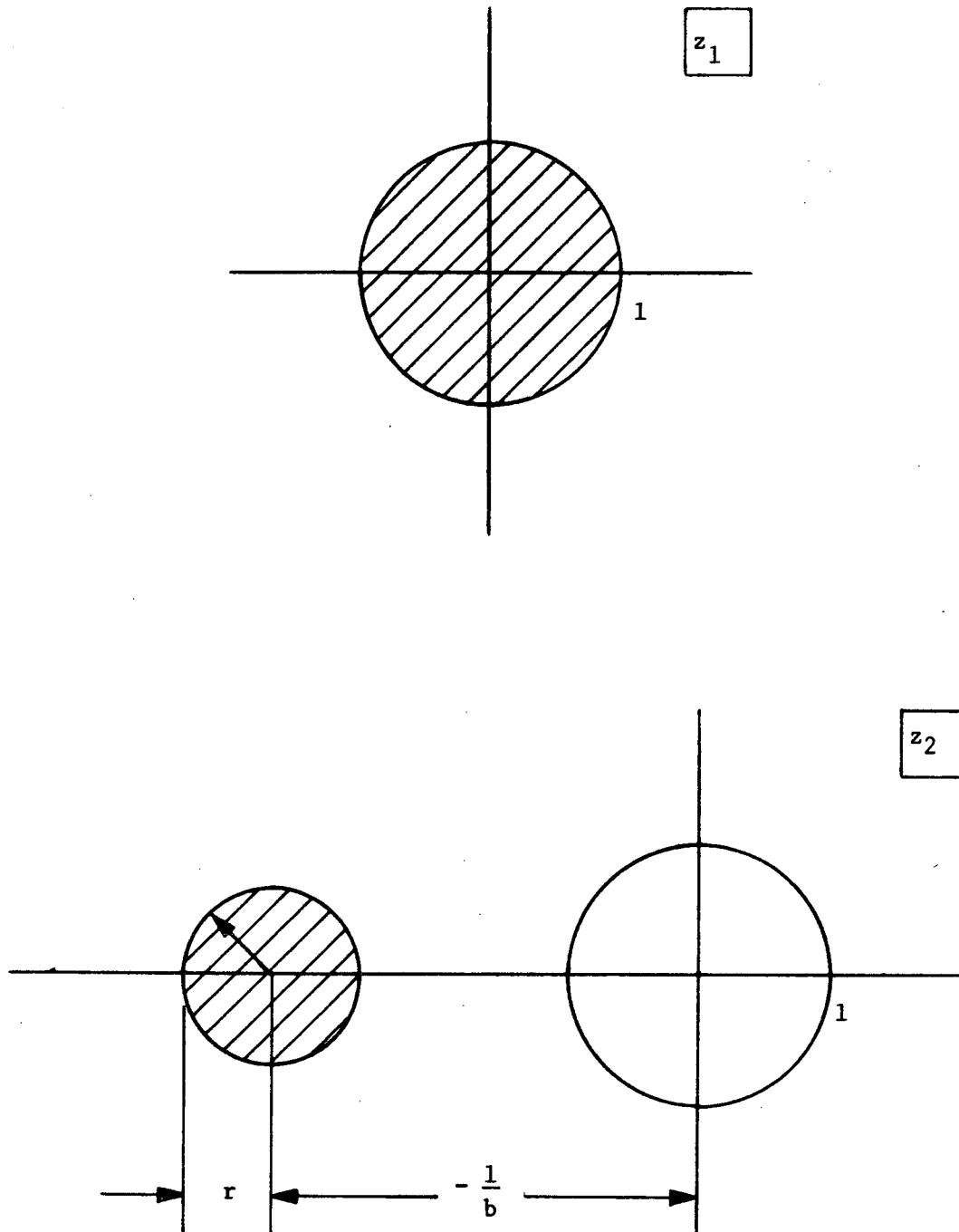


Fig. 14. Stability in Two-Dimensional Digital Filters.

or

$$|a| + |b| < 1$$

for stability.

Condition 2 checks the point $z_2 = 0$ in the z_1 plane:

$$0 = -\frac{1}{b} - \frac{a}{b} z_1$$

$$z_1 = -\frac{1}{a}$$

but z_1 must lie outside the unit circle, so

$$\left| -\frac{1}{a} \right| > 1$$

$$|a| < 1$$

which is included in condition 1. Therefore, the example filter is stable if the sum of the magnitudes of the coefficients is less than one.

Non-recursive two-dimensional digital filters may also be designed using windows, just as their one dimensional brothers. If $w_1(x)$ is a good one-dimensional window, then

$$w_2(x,y) = w_1(\sqrt{x^2 + y^2}) \quad (1-45)$$

will be a good two-dimensional window function.

Example. Consider the one-dimensional window

$$\begin{aligned} w(x) &= 1 - |x| & |x| &\leq 1 \\ &= 0 & |x| &> 1 \end{aligned}$$

Then

$$W(\omega) = \sin^2(\omega/2)/(\omega^2/4)$$

which has sidelobes of about 4%.

The two-dimensional counterpart is

$$\begin{aligned} w_2(x,y) &= 1 - \sqrt{x^2 + y^2} & |x^2 + y^2| &\leq 1 \\ &= 0 & |x^2 + y^2| &> 1 \end{aligned}$$

Then, in the frequency domain

$$W_2(\omega_1, \omega_2) = 2\pi[\rho^{-3} \int_0^\rho J_0(t) dt - \rho^2 J_0(\rho)]$$

$$\rho = \sqrt{\omega_1^2 + \omega_2^2}$$

which has sidelobes of only 2%.

The reader is referred to the open literature where much two-dimensional digital filtering theory is reported.

Adaptive Digital Filters [20]

A major advantage which digital filters hold over analog ones is the ease in which a digital filter's coefficients may be changed while the filter is processing data. Adaptive digital filters change their coefficients to minimize some specified criteria. An example non-recursive adaptive digital filter is depicted in Fig. 15a. The filter output is

$$\hat{d}(iT) = \sum_{k=0}^K g_k^{(i)} x(iT - kT) \quad (1-46)$$

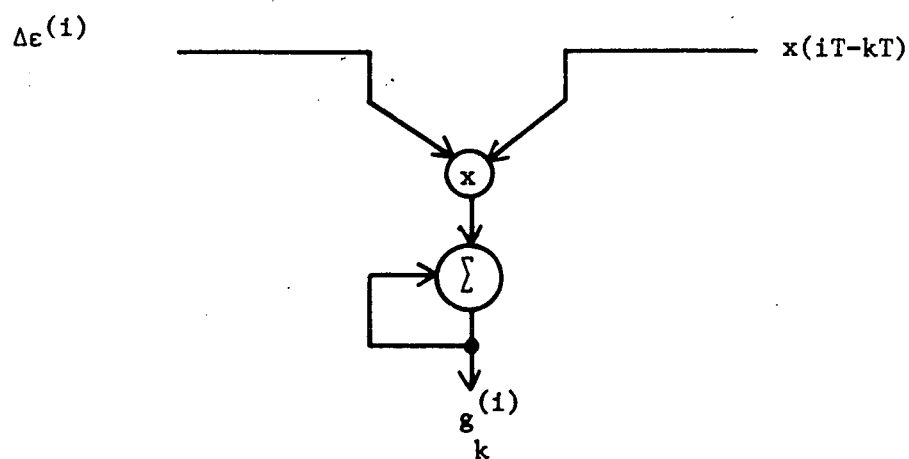
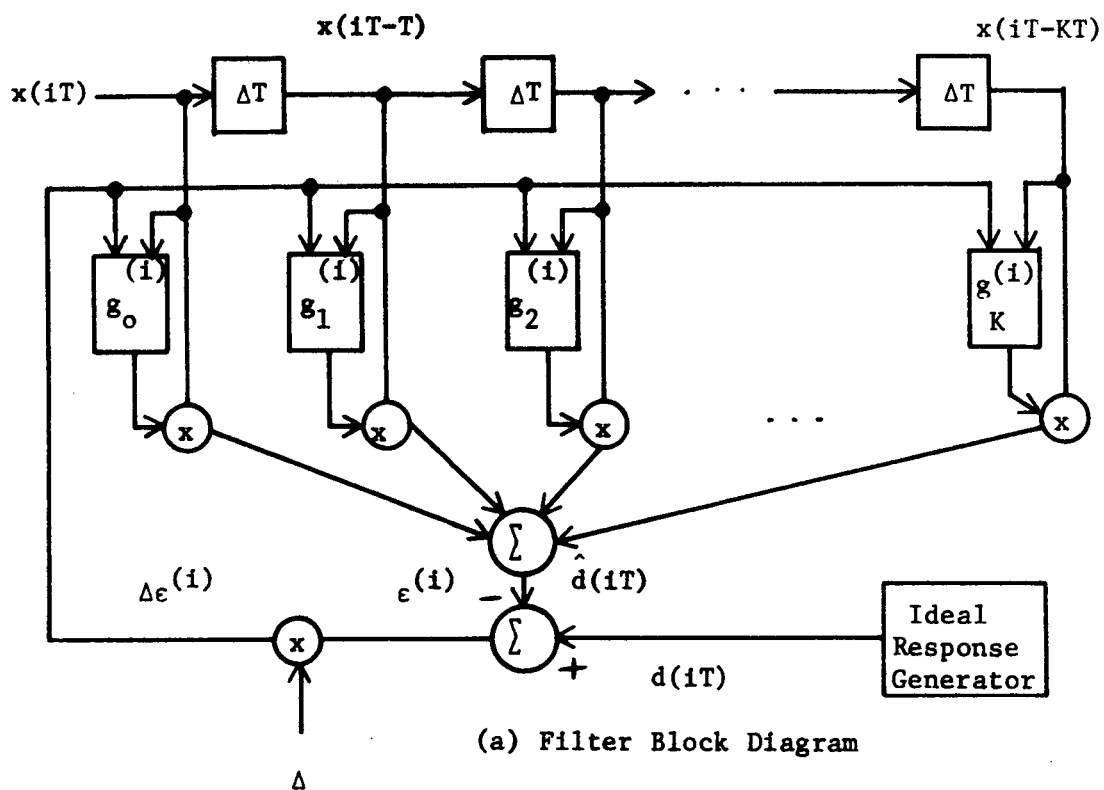
where $g_k^{(i)}$ are time varying coefficients calculated as shown in Fig. 15b.

$$g_k^{(i+1)} = g_k^{(i)} + \Delta \epsilon^{(i)} x(iT - kT). \quad (1-47)$$

The term $\epsilon^{(i)}$ is found by subtracting the filter response from an ideal response $d(iT)$. The factor Δ is a variable step length which is adjusted to improve the filter response in driving $\epsilon^{(i)}$ toward zero.

Floating Point Digital Filters

A floating point digital filter is one which is implemented by a computing device which executes floating point arithmetic in calculating the filter's difference equations. Both the filter's coefficients and the signal variables are represented in the following format



(b) Time Varing Gain Generation

Fig. 15. An Adaptive Digital Filter.

$$F \times R^E$$

(1-48)

where F is a fraction expressed in radix R and E is the exponent value. R is usually 2 in 16-bit minicomputers but is 16 in IBM 360/370 machines. Digital filters are not usually implemented in floating point for several reasons. Floating point hardware is slower than fixed point and is more costly. Perhaps a more important reason is that floating point quantization errors in signal variables can cause system instability whereas with fixed point arithmetic is guaranteed to be stable if the filter coefficients yield stable poles in the z -plane.

Optimal Digital Filtering

Optimal digital filters are filters used to minimize some performance evaluation criterion set for the discrete filter. In this section, three topics will be presented: 1) the concept of optimization, 2) the optimal control law, and 3) state estimation.

Concept of Optimization [21]. A system may be described by n first order linear or non-linear differential equations in the independent variables x_1, x_2, \dots, x_n . Any system can be so described by the introduction of the appropriate number of variables, henceforth referred to as the state variables. The n differential equations are

$$\dot{\underline{x}} = \underline{f}(\underline{x}, \underline{u}, t)$$

(1-49)

Suppose that a function

①

$$V(\underline{u}) = \int_0^{mT} L(\underline{x}, \underline{u}, t) dt \quad (1-50)$$

is to be minimized by choosing the forcing functions $\underline{u}(t)$ or some other system parameters. L represents the performance criterion together with any terms which penalize or restrict the use of forcing signals. The minimum value of $V(\underline{u})$ is termed the cost.

A linear optimal system has the following characteristics:

- (a) linear differential equations
- (b) the performance criterion has a quadratic form in the state variables and forcing functions
- (c) unrestricted forcing functions and state variables.

Any system which does not possess all three characteristics is non-linear.

Consider the linear system described by the following set of first order differential equations:

$$\dot{\underline{x}} = \underline{F}\underline{x} + \underline{G}\underline{u} \quad (1-51)$$

$$\underline{y} = \underline{H}\underline{x} \quad .$$

Now it is desired to calculate $\underline{u}(t)$ (given the initial values $\underline{x}(0)$) such that the cost function $V(\underline{u})$ is minimized.

It is proposed to approximate the system by a discrete time version. The time interval is divided into m equal sub-intervals T and the forcing function \underline{u} is to be held constant during each subinterval. The system

is considered to be described by a sequence of transitions from the (k-1)th to the (k)th state.

Solving the set of first order differential equations, we find the following transition equation:

$$\underline{x}(kT + T) = \phi(kT + T, kT)\underline{x}(kT) + \Gamma(kT + T, kT)\underline{u}(kT) \quad (1-52)$$

where

$$\Gamma(kT + T, kT) \triangleq \int_{kT}^{kT+T} \phi(kT + T, T)G(T) dT.$$

and $\phi(t, T)$ is found as follows:

1. When F is time varying, ϕ is computed from

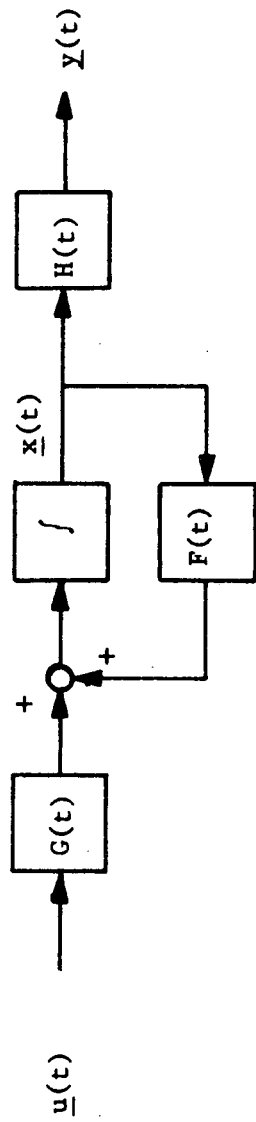
$$\frac{d[\phi(t, \tau)]}{dt} = F(t) \phi(t, \tau)$$

2. When F is constant $\phi(t, T) = \phi(t-T)$ is computed by

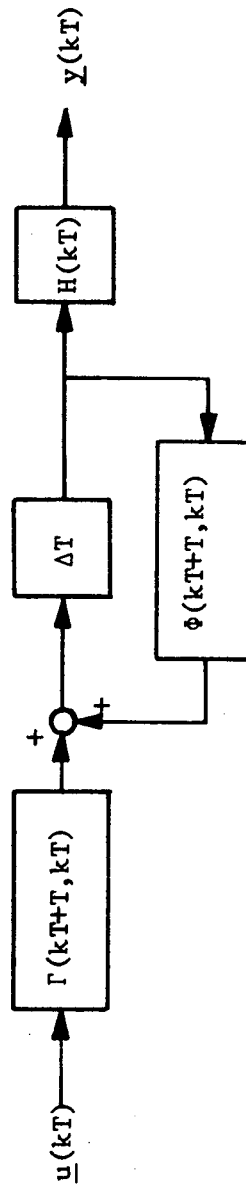
$$\phi(t-T) = e^{F(t-T)} = \sum_{k=0}^{\infty} \frac{[F(t-t_0)]^k}{k!}.$$

The relationship between continuous and discrete systems is shown in Fig. 16.

In addition, the integral to be minimized is replaced by the summation



(a) Continuous System



(b) Discrete Model of System

Fig. 16. The Continuous and Discrete Systems

$$V(mT) = T \sum_{k=0}^{m-1} L[\underline{u}(kT), \underline{x}(kT), kT] \quad (1-53)$$

The minimization of $V(mT)$ for discrete systems will be considered for two cases: a) the optimal control law, and b) state estimation.

Optimal Control Law [22]. Consider the continuous system equations to be of the form,

$$\dot{\underline{x}}(t) = F(t) \underline{x}(t) + G(t) \underline{u}(t)$$

$$\underline{y}(t) = H(t) \underline{x}(t) \quad (1-54)$$

$$V(mT) = \underline{x}^T(mT) A \underline{x}(mT) + \int_0^{mT} \underline{x}^T(t) B \underline{x}(t) dt + \int_0^{mT} \underline{u}^T(t) C(t) \underline{u}(t) dt$$

where A = terminal state weighting matrix

$B(t)$ = state weighting matrix

$C(t)$ = control cost matrix.

The optimal controller is obtained by solving the nonhomogenous matrix Riccati equation

$$\frac{dS}{dt} = -SF - F^T S + SGC^{-1}(0)C^T S - B(0). \quad (1-55)$$

If F and G are constant,

$$S(mT) = [\phi_{21}(mT) + \phi_{22}(mT)A][\phi_{11}(mT) + \phi_{12}(mT)A]^{-1} \quad (1-56)$$

Where

$$e^{MmT} \underline{\Delta} \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix}$$

and

$$M = \begin{bmatrix} -F & GC^{-1}G^T \\ B(0) & F^T \end{bmatrix} .$$

Once $S(mT)$ is known, the optimal control vector can be obtained from

$$\underline{u}_{opt}(t) = D(mT - t)\underline{x}(t) \quad (1-57)$$

where

$$D(mT - t) = C^{-1}(t)G^TS(mT - t) .$$

In block diagram form, the optimum controller can be depicted as in Fig. 17. Notice that to find \underline{u}_{opt} the state vector $\underline{x}(t)$ is necessary for calculation of the optimal input. In most systems $\underline{x}(t)$ is not available; $\underline{y}(t)$ is available instead. Hence, we "estimate" $\underline{x}(t)$ using $\underline{y}(t)$ as shown in Fig. 17.

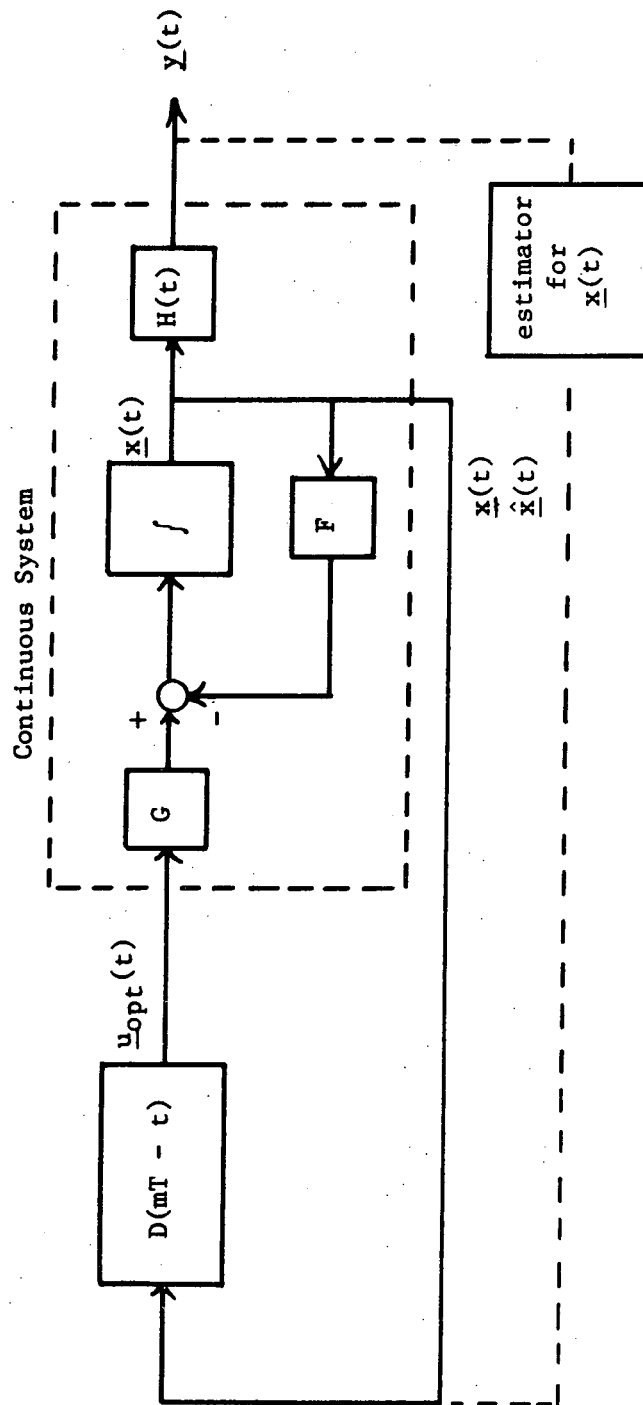


Fig. 17. Optimal Control System

State Estimation [23]. It is desired to find an optimal estimate $\hat{\underline{x}}$ for the state variables \underline{x} for a system defined in Fig. 17. The system output \underline{y} is measured every T seconds; call the measurement

$$\begin{aligned}\underline{z}(nT) &= \underline{y}(nT) + \underline{v}(nT) \\ &= H(nT) \underline{x}(nT) + \underline{v}(nT).\end{aligned}$$

simplifying the notation

$$\underline{z}_n = H_n \underline{x}_n + \underline{v}_n \quad (1-58)$$

where \underline{v}_n is measurement noise and

$$\begin{aligned}E[\underline{v}_{n-m} \underline{v}_m^T] &= R_n \delta_{mn} \\ E[\underline{v}_n] &= 0.\end{aligned} \quad (1-59)$$

The estimation scheme is to predict the present value of the state vector by using the last predicted value and updating it with the present measurement.

$$\hat{\underline{x}}_n(+) = \hat{\underline{x}}_n(-) + K_n [\underline{z}_n - H_n \hat{\underline{x}}_n(-)] \quad (1-60)$$

where $\hat{\underline{x}}_n(+)$ and $\hat{\underline{x}}_n(-)$ are estimates of the state vector \underline{x}_n after

and before the measurement z_n , at time nT . The K_n is the optimum weighting matrix. Let the error in the estimate be

$$\begin{aligned}\tilde{x}_n(+) &= \hat{x}_n(+) - x_n \\ \tilde{x}_n(-) &= \hat{x}_n(-) - x_n\end{aligned}\tag{1-61}$$

Substituting (1-58) and (1-61) into (1-60)

$$\tilde{x}_n(+) = (I - K_n H_n) \tilde{x}_n(-) + K_n v_n, \tag{1-62}$$

Define

$$P_n(+) \triangleq E[\tilde{x}_n(+) \tilde{x}_n^T(+)] \tag{1-63}$$

However,

$$E[x_n(-) v_n^T] = E[v_n x_n^T(-)] = 0$$

because of uncorrelated measurement errors. Thus, (1-63) becomes

$$P_n(+) = (I - K_n H_n) P_n(-) (I - K_n H_n)^T + K_n R_n K_n^T. \tag{1-64}$$

The cost function to be minimized in state estimation is the sum of the diagonal elements of the error covariance matrix $P_n(+)$:

$$V(mT) = \sum_{n=0}^{m-1} \{ E[\tilde{x}_n^T(+)\tilde{x}_n(+)] \}$$

The $V(mT)$ is minimized by K_n . The solution is

$$K_n = P_n(-)H_n^T[H_nP_n(-)H_n^T + R_n]^{-1} = P_n(+)^T R_n^{-1} \quad (1-65)$$

Substituting K_n into $P_n(+)$ results in

$$P_n(+) = P_n(-) - P_n(-)H_n^T[H_nP_n(-)H_n^T + R_n]^{-1} H_nP_n(-). \quad (1-66)$$

The equation set for the state transitions of the discrete system

$$\underline{x}_{n+1} = \phi_n \underline{x}_n + \underline{w}_n. \quad (1-67)$$

Again, using (1-67) and (1-61) in (1-63)

$$P_{n+1}(-) = \phi_n P_n(+) \phi_n^T + Q_n, \quad (1-68)$$

where

$$\underline{w}_n = \Gamma[kT + T, kT] \underline{u}(kT)$$

$$E[\underline{w}_{n-m} \underline{w}_n^T] = Q_n \delta_{mm}$$

$$E[w_n] = 0.$$

The results given above are now summarized in Fig. 18.

Nonlinear Filtering [24]

Reference [24] presents a class of nonlinear systems which obey a principle of superposition. In particular, the synthesis of nonlinear filters for signals which can be expressed as a product or convolution of components is examined. Practical applications in speech and image processing are illustrated.

Range Adaptive Digital Filtering [25]

In many applications the digital filter's input signal tends to dwell near zero with occasional perturbations away from null. Range adaptive digital filtering has automatic scaling of its input, internal, and output signals to prevent arithmetic overflow. This is a hardware concept and will be further examined in PART 3, Mechanization of Digital Filters.

Random Sample Skipping [26]

In certain time-shared applications of digital filter hardware several input/output sequences, say n , of numbers can be handled by a single special-purpose computer which looks like n digital filters. If the sampling rates for each filter is different, then inevitably conflicts for the arithmetic unit will take place and certain input samples will essentially be lost. This process can be described as

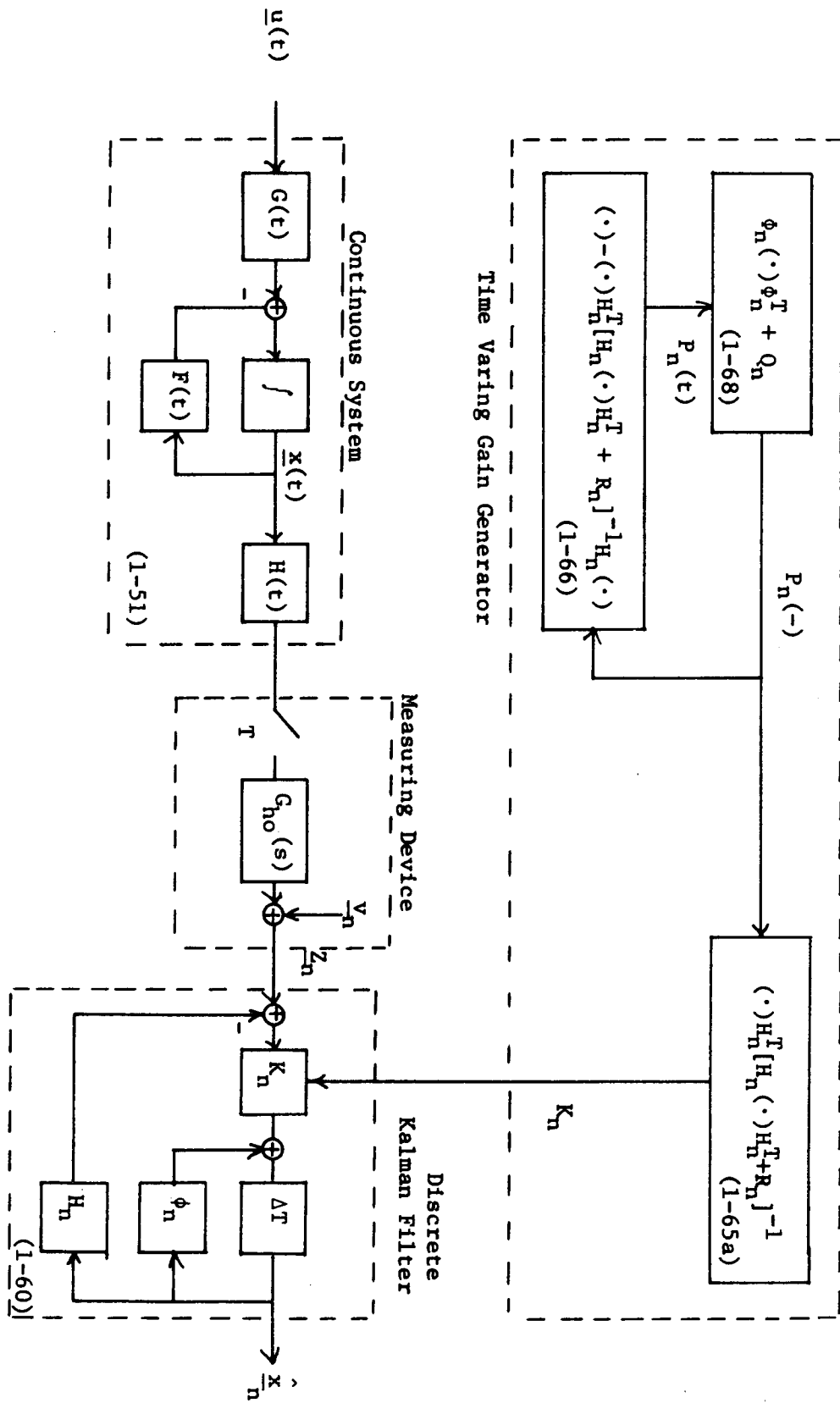


Fig. 18. The Continuous System with Optimal State Estimation

nonlinear random sample omission. Reference [26] shows that in some cases random sample omissions in a closed-loop system with random inputs can be beneficial in reducing the mean-square value of a nulling error signal.

Block-Floating-Point Filters [27]

Block-floating-point is a compromise between fixed-point and floating-point arithmetic. In fixed point arithmetic no scaling is used for addition or multiplication of numbers. In floating-point, automatic scaling is performed for each product or sum calculated. In block-floating point arithmetic, numbers are expressed as a fraction and exponent (as in floating point); however, scaling is performed once for an entire expression instead of for each operation.

For example,

$$y_n = x_n + a_1 y_{n-1} + \dots + a_N y_{n-N} \quad (1-69)$$

would be calculated as

$$\begin{aligned} y_n &= \frac{1}{A_n} \hat{y}_n \\ \hat{y}_n &= \Delta_n \hat{x}_n + a_1 \Delta_n w_{1n} + \dots + a_N \Delta_n w_{Nn} \\ \hat{x}_n &= A_{n-1} x_n \\ w_{in} &= A_{n-1} y_{n-i} \end{aligned} \quad (1-70)$$

The scaling factors A_n and Δ_n are powers of 2 and are determined as follows

$$\Delta_n = \frac{1}{2^{C_n}}$$

$$A_n = \Delta_n A_{n-1}$$

where C_n is the maximum characteristic of the variables $\hat{x}_n, w_{1n}, \dots, w_{Nn}$. In (1-70), the calculations for y_n, \hat{x}_n , and w_{in} involve only scaling (shifting). Once scaling is performed in \hat{y}_n , then all the arithmetic calculations are performed in fixed-point. The block-floating-point realization is summarized in Fig. 19.

Sample-Rate Reduction Digital Filters [28]

There exists a direct relation between input sampling frequency and the computational rate of the digital filter hardware implementation. In order to prevent input frequency aliasing, two common practices are to sample at a high rate or to use an analog low-pass filter before the A/D converter. Reference [28] suggests sampling at a high rate and using a digital low-pass filter whose output can be sampled at a much lower rate to furnish the input signal for some digital signal processing system. Advantages include the elimination of phase distortions which are inevitable in analog aliasing filters.

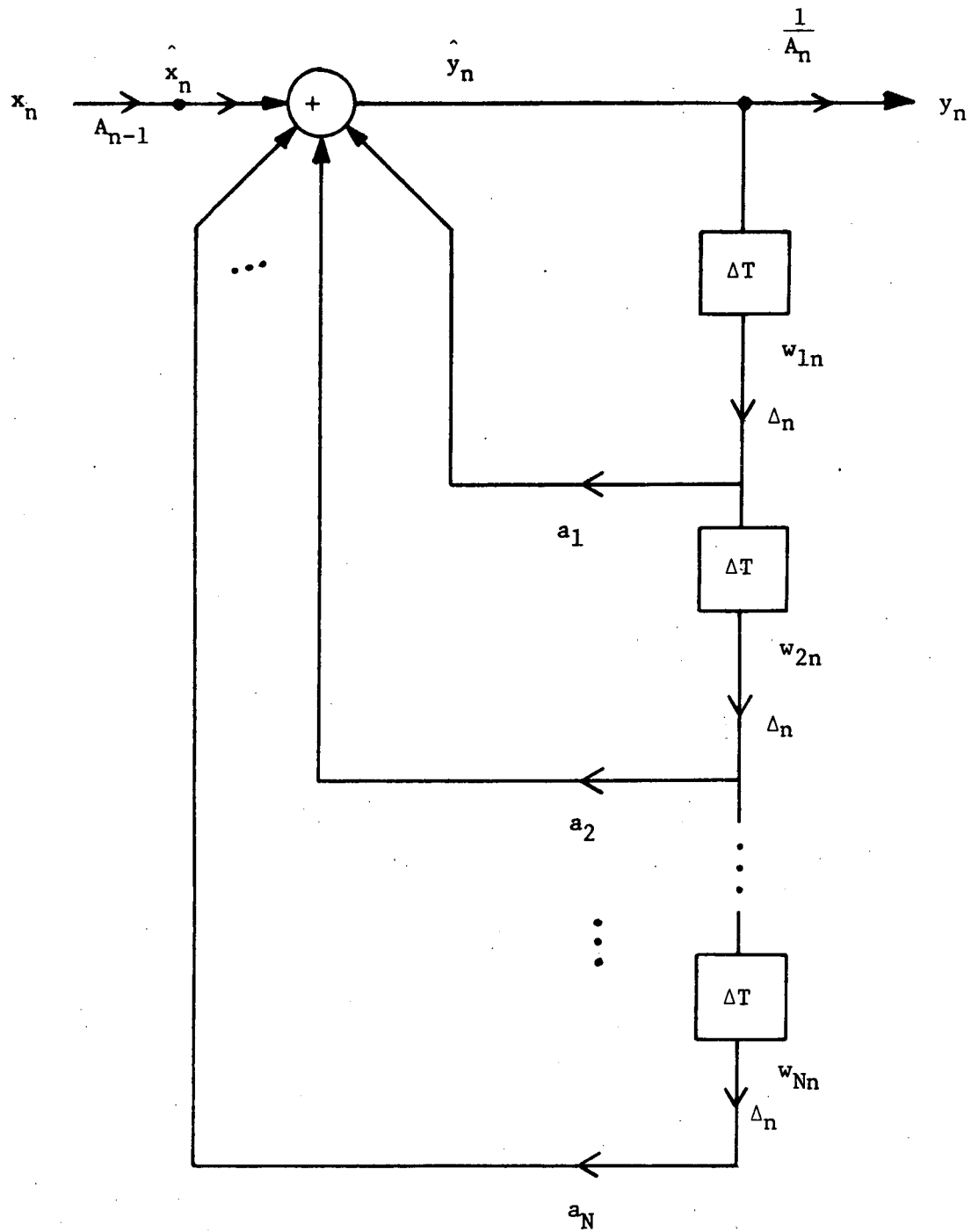


Fig. 19. Block-Floating-Point Filter

II. TRANSFER FUNCTION SYNTHESIS

The synthesis of transfer functions for digital filters is surveyed in this section. The survey is subdivided into nonrecursive filters, recursive filters, and sample designs.

Nonrecursive Filters

The synthesis of nonrecursive digital filters consists of determining the coefficients h_1 of the expression

$$H(z) = \sum_{i=0}^{M-1} h_i z^{-i} . \quad (2-1)$$

In the z -plane this amounts to placing zeroes anywhere in the plane with all poles falling at the origin.

Specification of Frequency-Domain Zeroes [29]

The design of nonrecursive digital filters in the frequency domain consists of specifying a finite trigonometric polynomial which satisfies some criteria. Here the polynomial is defined by placing its zeroes.

The frequency characteristic of (2-1) is defined as

$$H(f) = \sum_{n=0}^{M-1} h_n e^{-j2\pi fn} .$$

Replacing f by the complex variable $\phi = f + j\sigma$

$$H(\phi) = K_1 \prod_{n=1}^{M-1} [1 - e^{-j2\pi(\phi - \phi_n)}] \quad (2-2)$$

where

$$\{\phi_n\} = \{f_n + j\sigma_n\}$$

$$|f_n| \leq 1/2$$

The $\{\phi_n\}$ are the zeroes of $H(\phi)$ in the central period. Hence, the scale factor K_1 and the $M-1$ central zeroes completely specify $H(f)$.

The function is factored into stopband and passband zeroes

$$H(f) = H_S(f)H_P(f) \quad (2-3)$$

where

$$H_S(f) = K_S \prod_{n=1}^{N_S} [1 - e^{-j2\pi(f - \phi_n)}]$$

$$H_P(f) = K_P \prod_{n=1}^{N_P} [1 - e^{-j2\pi(f - \psi_n)}] \quad (2-4)$$

Once the zeroes have been apportioned between the stopband and passband, the passband and stopband zeroes are positioned to give a "good" shape for $H(f)$. The procedure is demonstrated in [29].

Frequency Sampling [30,31,32]

The technique of frequency sampling may be used to synthesize nonrecursive filters as follows:

1. Choose a set of frequencies at which the sampled frequency response is specified.
2. Obtain the values of the continuous frequency response of the resulting filter as a function of the filter parameters (defined below) using the sampling theorem.
3. Compare the interpolated frequency response with the desired filter and search for a minimum of some filter characteristic.
4. When the minimum is found, the parameters are used to realize the nonrecursive filter.

The frequency samples in step 1 are specified in the passband and stopband; however, in the transition region several samples are left adjustable and these are the parameters used in steps 2 and 3. The references [30,31] describe computer aided design programs which essentially automate the optimizing process.

Windowing [19,31,33]

Windowing filters, discussed in Chapter 1, are nonrecursive filters whose finite impulse response is found by terminating an

infinite impulse response by means of a window function. The details of the procedure have been demonstrated earlier.

Equiripple Filters [34,35]

Equiripple nonrecursive digital filters may be designed by minimizing the maximum error between some desired complex frequency response $F(f)$ and the FIR response as shown below

$$E_k = \sum_{\ell=-P/2}^{P/2} h_{\ell} e^{-2\pi j f_k \ell} - F(2W f_k) \quad (2-5)$$

where

$$j = \sqrt{-1}$$

h_{ℓ} = filter coefficients

P = even integer

f_k = normalized sampled frequency

$$|f_k| \leq 1/2$$

$2W$ = sampling rate.

Equation (2-5) may be minimized using the simplex method of linear programming. Digital filters designed in this manner are sometimes said to have minimax responses.

Recursive Filters

The recursive digital filter has the form

$$H(z) = \frac{\sum_{i=0}^n a_i z^{-i}}{1 + \sum_{i=1}^n b_i z^{-i}} \quad (2-6)$$

The synthesis of recursive filters is the task of choosing the coefficients a_i and b_i in order to force the filter to behave in some specified manner.

Direct Synthesis in the Frequency Domain [36]

The frequency response for (2-6) is found by substituting $z = e^{j2\pi fT}$

$$H(f) = \frac{\sum_{i=0}^n a_i e^{-j2\pi f i T}}{1 + \sum_{i=1}^n b_i e^{-j2\pi f i T}} \quad (2-7)$$

If $N(f)$ is the numerator of (2-7), then

$$\begin{aligned} |N(f)|^2 &= \left(\sum a_i e^{-j2\pi f i T} \right) \left(\sum a_i e^{j2\pi f i T} \right) \\ &= H_0 + 2 \sum_{k=1}^n H_k \cos(2\pi k T f) \end{aligned} \quad (2-8)$$

where

$$H_0 = \sum_{k=0}^n k^2$$

$$H_k = \sum_{(p-q)=k} a_p a_q .$$

Equation (2-8) may be further reduced to

$$|N(f)|^2 = \sum_{k=0}^n \alpha_k \cos^{2k}(\pi T f), \quad (2-9)$$

where α_k are constants. Hence equation (2-7) may be written as a rational function in $\cos(\pi T f)$ [or $\sin(\pi T f)$]. Any such rational function may be specified by the roots of the two polynomials.

Consider the Butterworth lowpass filter in the analog domain

$$|H_1(f)|^2 = \frac{1}{1 + \left(\frac{f}{f_c}\right)^{2p}} .$$

Since the term $\sin(\pi f T)$ corresponds to f in the discrete case

$$|H_2(f)|^2 = \frac{1}{1 + \left[\frac{\sin(\pi f T)}{\sin(\pi f_c T)} \right]^{2p}} \quad (2-10)$$

represents a lowpass digital filter. To find the filter coefficients solve for the roots of the polynomial. An example design in the continuous case is presented later in this chapter.

Other filter types (bandpass, highpass, band stop, etc) may be designed using this technique.

Sampled Data Transformations [37]

This section describes a mapping technique for designing recursive digital filters. First, a suitable continuous filter $G(s)$ is found, and then a mapping function from the s -plane to z -plane is employed to find the digital equivalent filter $D(z)$. Hence, first we review continuous filter design and then employ the sampled-data transformations.

Continuous Filter Design. The design of continuous filters can be accomplished by first designing several low pass filter transfer functions $G(s)$, called prototype or normalized designs; the prototypes have a critical or break frequency of one radian/sec. The prototype is used to realize a filter for a given specification by using the frequency transformations listed below:

$$\begin{aligned}
 \text{Low Pass:} \quad s &\rightarrow s/\omega_u \\
 \text{Band Pass:} \quad s &\rightarrow \frac{s^2 + \omega_u \omega_\ell}{s(\omega_u - \omega_\ell)} \\
 \text{Band Stop:} \quad s &\rightarrow \frac{s(\omega_u - \omega_\ell)}{s^2 + \omega_u \omega_\ell} \\
 \text{High Pass:} \quad s &\rightarrow \omega_u/s
 \end{aligned} \tag{2-11}$$

where

ω_u = upper cutoff

ω_l = low cutoff

Five prototype filters will be discussed in this section:

Butterworth, Bessel, Transitional, Chebyshev, and Elliptic designs.

Butterworth: The Butterworth approximation to the ideal low pass filter is defined by the squared frequency magnitude function

$$|G(\omega)|^2 = 1/[1 + (\omega^2)^n] \quad (2-12)$$

where n is the order of the filter. The Laplace transfer function is given by

$$G(s) G(-s) = 1/[1 + (-1)^n s^{2n}] ,$$

or

$$G(s) = \prod_{j=1}^n \frac{1}{(s + b_j)}$$

where

$$b_j = -e^{i\pi[(1/2) + (2j-1)/2n]} \quad i = \sqrt{-1}$$

Bessel: The Bessel filter approximation for the linear delay function e^{-Ts} may be written

$$G(s) = \frac{K_0}{B_n(s)} \quad (2-13)$$

where K_0 is a constant term and $B_n(s)$ are Bessel polynomials.

$$B_0 = 1$$

$$B_1 = s + 1$$

$$\vdots$$

$$B_n = (2n-1) B_{n-1} + s^2 B_{n-2}$$

The roots of $B_n(s)$ are normalized using the factor $(K_0)^{1/n}$

Transitional: The transitional filter combines roots of the n^{th} order Butterworth and normalized Bessel filters according to a transitional factor TF. Let

r_j = magnitude of j^{th} transitional pole

r_{1j} = magnitude of j^{th} Bessel pole

θ_j = angle of j^{th} transitional pole

θ_{1j} = angle of j^{th} Bessel pole

θ_{2j} = angle of j^{th} Butterworth pole.

the poles of the transitional filter are then described by

$$r_j = r_{1j}^{TF}$$

$$\theta_j = \theta_{2j} + TF(\theta_{1j} - \theta_{2j}) \quad (2-14)$$

Chebyshev: Chebyshev filters exhibit better cutoff characteristics for lower order filters than do the above designs. Chebyshev type I and type II filters are defined by

$$|G_1(\omega)|^2 = \frac{1}{1 + \epsilon^2 T_n^2(\omega)} \quad (2-15)$$

and

$$|G_2(\omega)|^2 = \frac{1}{1 + \epsilon^2 \left[\frac{T_n(\omega_r)}{T_n(\omega_r/\omega)} \right]^2} \quad (2-16)$$

where

$$\begin{aligned} T_n(\omega) &= \cos(n \cos^{-1} \omega) & 0 \leq \omega \leq 1 \\ &= \cosh(n \cosh^{-1} \omega) & \omega > 1 \end{aligned}$$

$$T_0 = 1$$

$$T_1 = \omega$$

$$T_2 = 2\omega^2 - 1$$

$$T_3(\omega) = 4\omega^3 - 3\omega$$

The order of the filter n is determined by specifying inband ripple E and the lowest frequency at which a loss of a db is achieved. Hence,

$$\epsilon = (10^{E/10} - 1)^{1/2} \quad (2-17)$$

$$A^2 = 10^{a/10}$$

and

$$n = \frac{\cosh^{-1} \sqrt{A^2 - 1/\epsilon}}{\cosh^{-1}(\omega_r)} .$$

In equation (2-17), the variables E , a , or ω_r must be adjusted so the n will be an integer. The type I filter differs from the type II in that the type I exhibits equiripple in the pass band while type II has equiripple in the stop band.

Elliptic: The Elliptic filter has equiripple in both the pass and stop bands. Hence, this type design usually achieves the desired frequency response with a lower order n than any of the above types. The elliptic filter is determined by

$$|G(\omega)|^2 = \frac{1}{1 + \epsilon^2 \psi_n^2(\omega)} \quad (2-18)$$

where

$$\psi_n = \begin{cases} \operatorname{sn}\left[n \frac{K(k_1)}{K(k)} \operatorname{sn}^{-1}(\omega; k); k_1\right], & n \text{ odd} \\ \operatorname{sn}\left[K(k_1) + n \frac{K(k_1)}{K(k)} \operatorname{sn}^{-1}(\omega; k); k_1\right], & n \text{ even} \end{cases}$$

with

$$\chi = \int_0^\omega \frac{d\omega}{[(1-\omega^2)(1-k^2\omega^2)]^{1/2}} = \text{Elliptic integral of the first kind}$$

$\operatorname{sn}[\chi; k] = \omega = \text{Jacobian Elliptic function}$

$K(k) = \text{complete Elliptic integral of the first kind}$

$$= \int_0^{\pi/2} \frac{d\phi}{(1 - k^2 \sin^2 \phi)^{1/2}}$$

$$k = 1/\omega_r$$

$$k_1 = \epsilon(A^2 - 1)^{-1/2}$$

$$\epsilon = (10^{E/10} - 1)^{1/2}$$

$$A^2 = 10^{a/10}$$

where ϵ , a , ω_r were defined for the Chebyshev filter, the order n is found by

$$n = \frac{K(k_1')K(k)}{K(k_1)K(k')}$$

with

$$k' = (1 - k^2)^{1/2}$$

$$k_1' = (1 - k_1^2)^{1/2}$$

The result of any of the five design methods results in a Laplace transfer function $G(s)$ for the desired frequency response.

Sampled-Data Transformations. Once the continuous transfer function $G(s)$ has been determined, the transformation to the discrete or z -plane is made. Three methods of transformation will be presented: the standard z -transform, the bilinear z -transform, and the matched z -transform.

Standard z -transform: The problem of converting a continuous filter to a discrete one was presented earlier. It was shown that

$$\frac{E_o^*(s)}{E_i^*(s)} = G(z) = Z[G(s)]$$

and that

$$\frac{E_o(s)}{E_i^*(s)} = G_{ho}(s) Z[G(s)] .$$

But for small T

$$G_{ho}(s) \approx T$$

Hence,

$$\frac{E_o(s)}{E_i^*(s)} = TZ[G(s)] .$$

Define the digital filter $D(z)$ equivalent to $G(s)$ to be

$$D(z) = TZ [G(s)], \quad (2-19)$$

where $Z[G(s)]$ is the standard z-transform of $G(s)$. Hence,

$$D(z) = T \sum_{k=1}^n \frac{R_k}{1 - e^{-Tb_k} z^{-1}}$$

Note that the standard z-transform can be used only on bandlimited signals ($f < f_s/2$).

F

Bilinear z-transform: The bilinear z-transform may be used to obtain a discrete equivalent of $G(s)$ as follows:

$$D(z) = G'(s) \Big|_{s = (2/T)(1 - z^{-1})(1 + z^{-1})^{-1}} \quad (2-20)$$

Where $G'(s)$ is a continuous filter whose critical frequencies differ from $G(s)$ by

$$f'_c = 1/\pi T \tan(\pi f_c T). \quad (2-21)$$

Relation (2-21) is used before the continuous filter $G(s)$ is designed. The new filter $G'(s)$ is designed instead and then transformed to the z-plane by (2-20). The bilinear z-transform is a bandlimiting transformation with relatively flat magnitude characteristics in the pass and stop bands. However, the time response will be considerably different.

Matched z-transform: The matched z-transform matches the poles and zeroes of the discrete function to those of the continuous one. The digital equivalent of the $G(s)$ function is calculated as follows:

$$D(z) = G(s) \Big|_{\begin{array}{l} s + a_i = 1 - z^{-1}e^{-a_i T} \\ s + b_j = 1 - z^{-1}e^{-b_j T} \end{array}} \quad (2-22)$$

If $G(s)$ has no zeroes, it is sometimes necessary to multiply (2-22) by $(1 + z^{-1})^N$, N is an integer.

Summary: The standard z -transform is suitable for only bandlimited functions, while the bilinear and matched z -transforms are suitable for all filter types. The matched z -transform requires $G(s)$ in factored form; standard, in partial fraction form; and bilinear, in prewarped frequency form. The standard z -transform preserves the shape of the impulse-time response; the matched, the shape of the frequency response; and bilinear, the flat magnitude gain-frequency response characteristics. An example filter is designed and discretized in the following example.

Design Example. In this section a digital filter will be designed using the techniques summarized above.

Suppose it is desired to design a bandstop filter $G_1(s)$ with

$$\omega_u = 200 = 2\pi(31.831)$$

$$\omega_l = 170 = 2\pi(27.056).$$

Multiplied times this filter will be a low pass filter $G_2(s)$ with $\omega_n = 600 = 2\pi(95.493)$, with a d. c. gain of 1.356. The band stop filter will be designed from Butterworth, Bessel, and Chebyshev I prototypes with $n = 2$. The low pass filter will be designed with $n = 1$. The prototype of $G_2(s) = \frac{1}{s + 1}$.

The prototype filters for $G_1(s)$ are found below.

Butterworth: The Butterworth filter is defined by

$$|G_1(\omega)|^2 = \frac{1}{1 + \omega^4}$$

$$G_1(s)G_1(-s) = \frac{1}{1 + s^4}$$

$$G_1(s) = \frac{1}{(s - e^{i3\pi/4})(s - e^{i5\pi/4})}$$

$$G_1(s) = \frac{1}{s + \sqrt{2}s + 1}$$

Bessel: The Bessel prototype is defined by

$$G(s) = \frac{K_0}{B_2(s)} \quad \frac{3}{s^2 + 3s + 3}$$

Chebyshev I: The Chebyshev I filter is defined by

$$|G_1(\omega)|^2 = \frac{1}{1 + \epsilon^2 T_2^2(\omega)}$$

$$T_2(\omega) = 2\omega^2 - 1$$

$$\epsilon = (10^{E/10} - 1)^{1/2}$$

$$A^2 = 10^{a/10}$$

$$n = \frac{\cosh^{-1}(\sqrt{A^2 - 1/\epsilon})}{\cosh^{-1}(\omega_r)} = 2$$

Let $E = 1.33$ db, then

$$\epsilon = (10^{1.33} - 1)^{1/2} = .5$$

Let the filter gain be down 6 db at ω_r

$$a = 6$$

$$A^2 = 10^{.6} = 4.$$

$$n = \frac{\cosh^{-1}(\sqrt{A^2 - 1/\epsilon})}{\cosh^{-1}(\omega_r)} = 2$$

$$\cosh^{-1}(\omega_r) = \frac{1}{2} \cosh^{-1}(\sqrt{2}) = .44$$

$$\omega_r = 1.098$$

Hence,

$$|G_1(\omega)|^2 = \frac{1}{\omega^4 - \omega^2 + 1.25}$$

$$G_1(s) = \frac{1}{(s + 1.057 \angle 31.75^\circ)(s + 1.057 \angle -31.75^\circ)}$$

$$G_1(s) = \frac{1}{s^2 + 1.308s + 1.118}$$

The analog filters are designed from the prototypes by setting

$$G(s) = G_1(s) \quad \times \quad G_2(s)$$

$$s = \frac{s(\omega_u - \omega_l)}{s^2 + \omega_u \omega_l} \quad \left| \quad s = s/\omega_n \right.$$

and adjusting the d.c. gain to be 1.356. The resulting filter equations are given by

Butterworth:

$$G(s) = 813.6 \frac{s^4 + 68000s^2 + 1.156 \times 10^9}{(s^4 + 1272.8s^3 + 68900s^2 + 4.3275 \times 10^7 s + 1.156 \times 10^9)(s + 600)}$$

Bessel:

$$G(s) = 2440.8 \frac{s^4 + 68000s^2 + 1.156 \times 10^9}{(3s^4 + 27000s^3 + 2.049 \times 10^5 s^2 + 9.18 \times 10^7 s + 3.465 \times 10^9)(s + 600)}$$

Chebyshev I:

$$G(s) = 909.6 \frac{s^4 + 68000s^2 + 1.56 \times 10^9}{(1.118s^4 + 1177.2s^3 + 76924s^2 + 4.0025 \times 10^7 s + 1.2924 \times 10^9)(s + 600)}$$

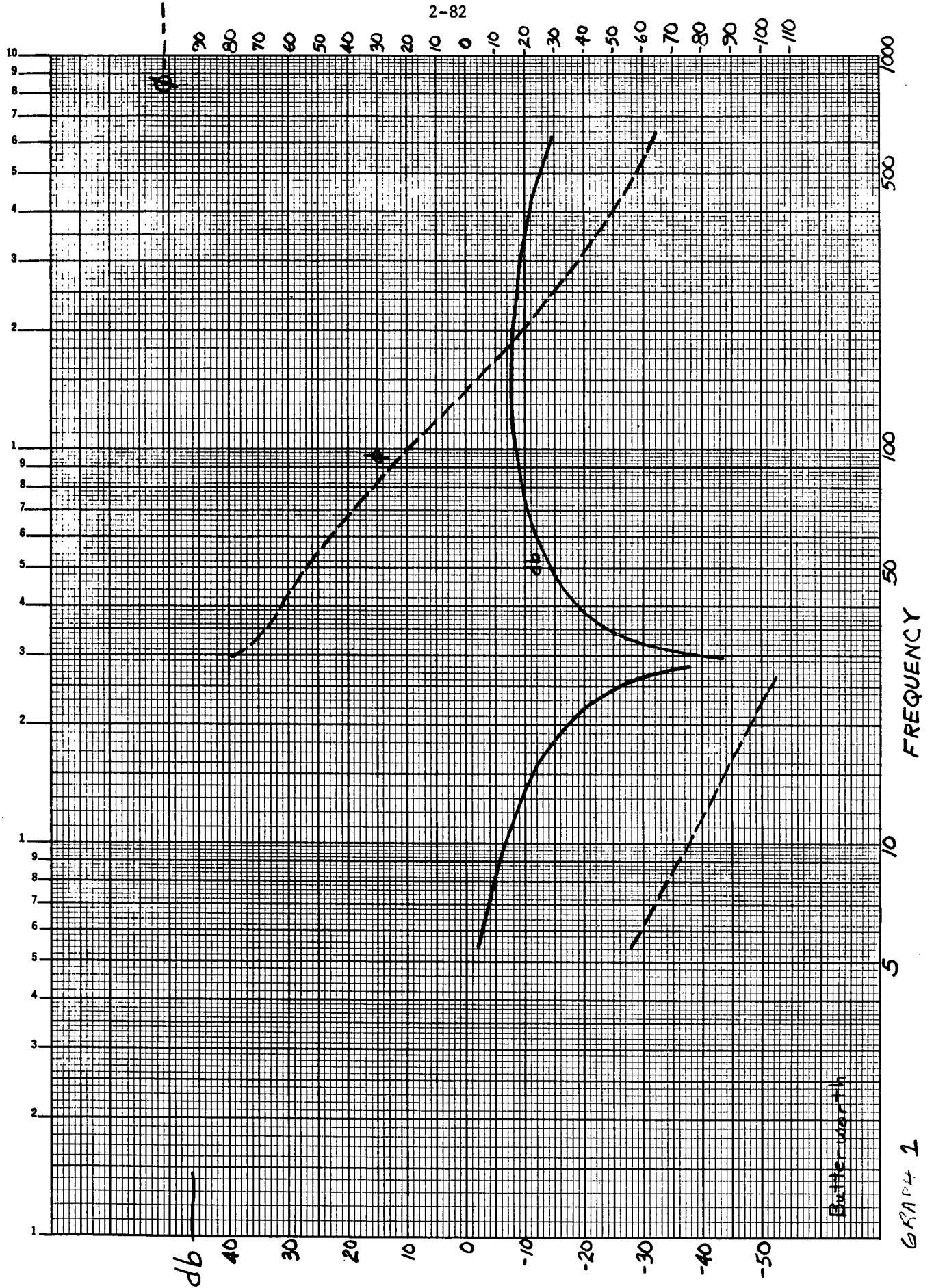
The filter equations above were plotted for db and phase, ϕ , versus frequency as shown in graphs 1, 2, and 3. Since the plots are nearly identical, the Butterworth $G(s)$ was chosen to be discretized by the standard, bilinear, and matched z-transforms, with $T = .001$.

The Butterworth design for $G(s)$ may be written in partial fraction expansion

$$G(s) = \frac{40.567}{s + 27.314} + \frac{2.4402 \times 10^{-4} + j7.5033 \times 10^{-5}}{x + .35375 + j185.39} \\ + \frac{2.5502 \times 10^{-4} - j7.5033 \times 10^{-5}}{x + .35375 - j185.39}$$

$$\frac{1}{S^2 + \sqrt{2}S + 1}$$

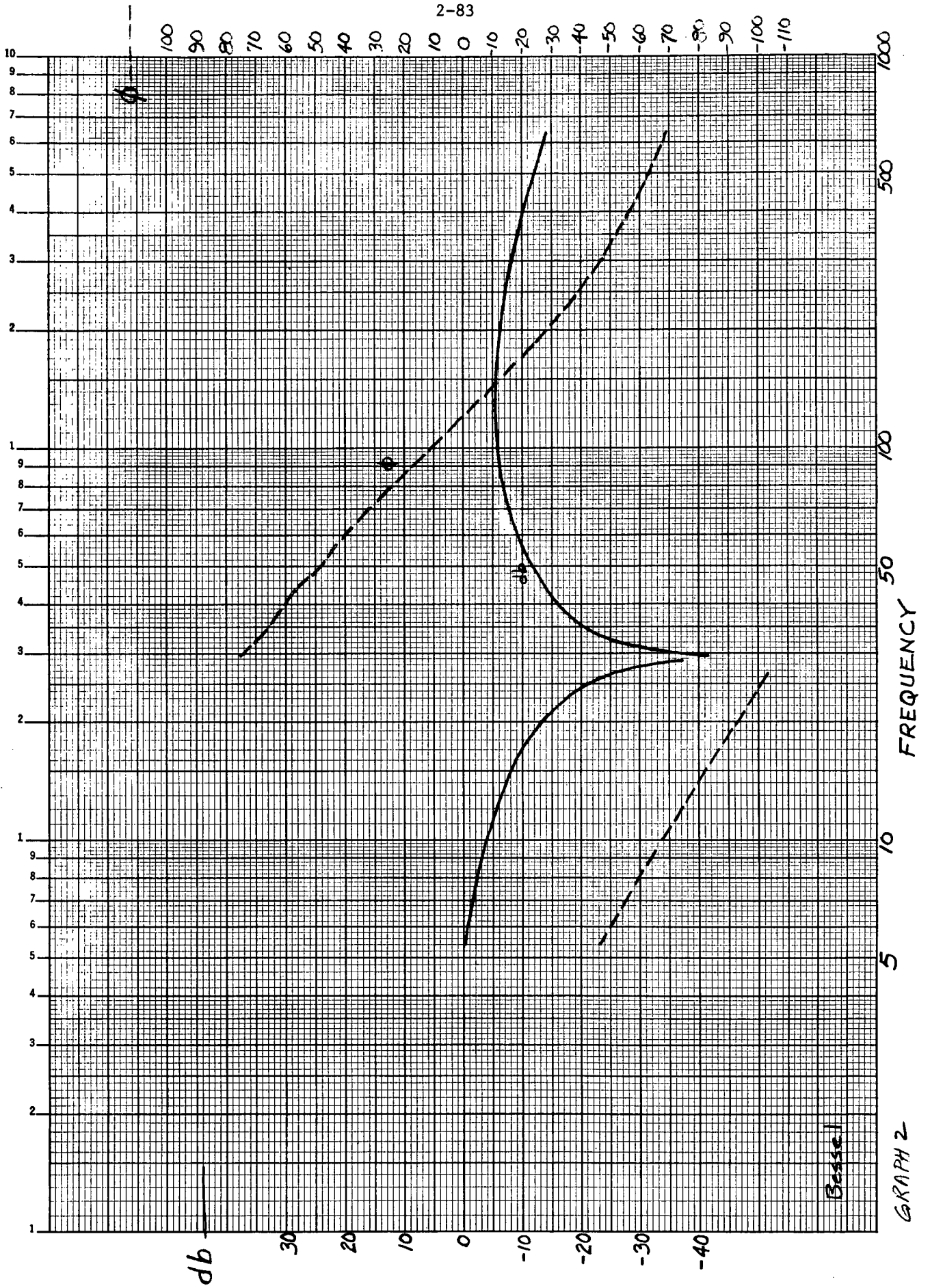
2-82



GRAPH 1

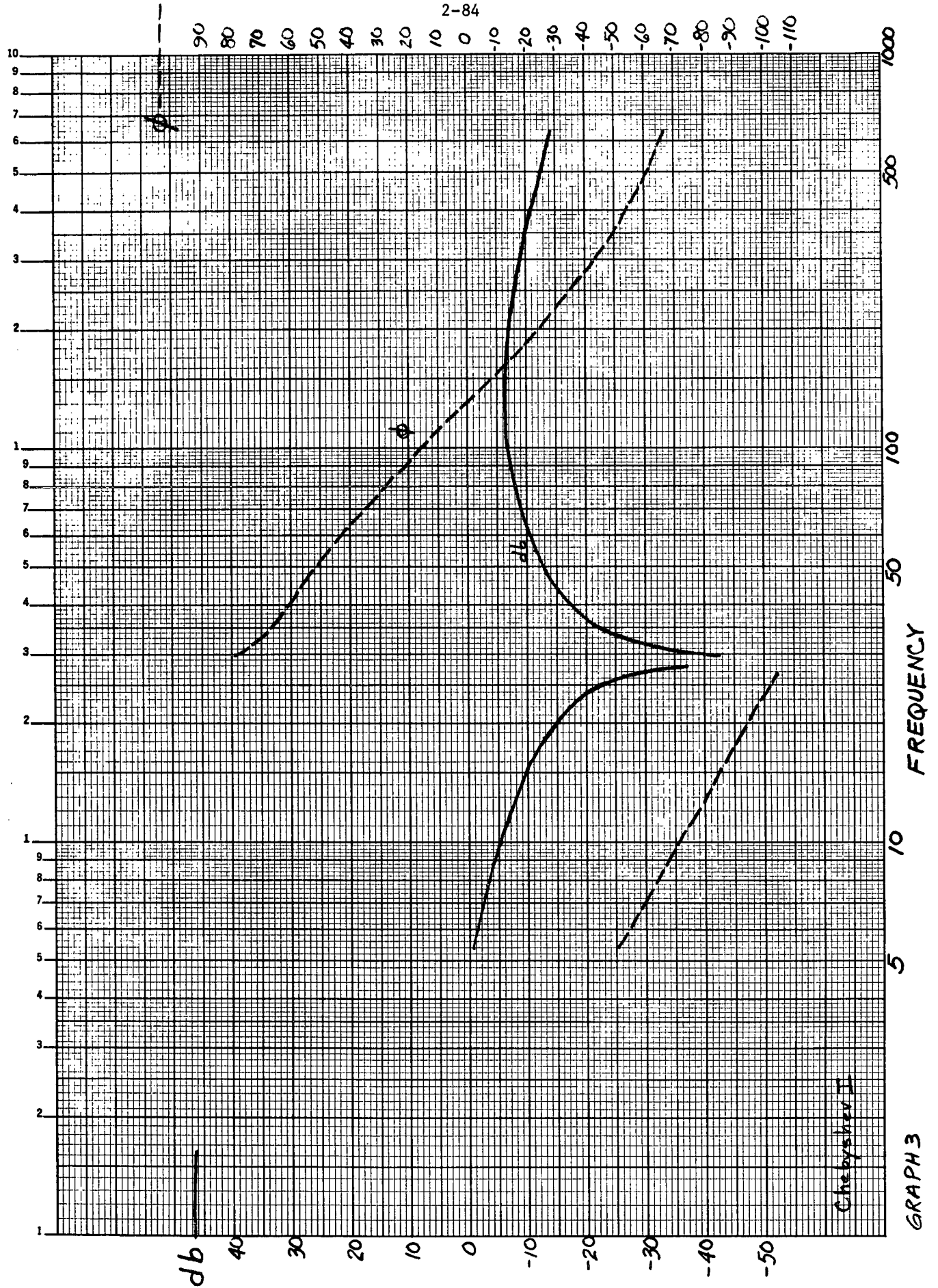
K&E SEMI-LOGARITHMIC 46 5813
 5 CYCLES X 140 DIVISIONS
 MADE IN U.S.A.
 KRUPP & ESSER CO.

$$S^2 + 3S + 3$$



$$\frac{1}{s^2 + 1.30s + 1.113}$$

2-84



$$+ \frac{1642.1}{s + 1244.8} + \frac{-869.03}{s + 600} .$$

The standard z-transform is taken

$$\frac{a}{s + u} \rightarrow \frac{aT}{1 - e^{-uT}z^{-1}}$$

and

$$\frac{a + ib}{s + u + iv} + \frac{a - ib}{s + u - iv} \rightarrow T \frac{[2a] + [2e^{-uT}(b\sin vT - a\cos vT)]z^{-1}}{1 + [-2e^{-uT}\cos vT]z^{-1} + [e^{-2uT}]z^{-2}}$$

Hence,

$$D(z) = \frac{4.0567 \times 10^{-2}}{1 - .97306z^{-1}} + \frac{4.8803 \times 10^{-7} - 4.420 \times 10^{-7}z^{-1}}{1 - 1.9654z^{-1} + .99929z^{-2}} \\ + \frac{1.6421}{1 - .28800z^{-1}} + \frac{-.86903}{1 - .54881z^{-1}}$$

The frequency response of this function is found by letting $z = e^{j\omega T}$.

The plot is shown in graph 4. Note that this response is entirely

inadequate. The standard z-transform is accurate only when $G(s)$

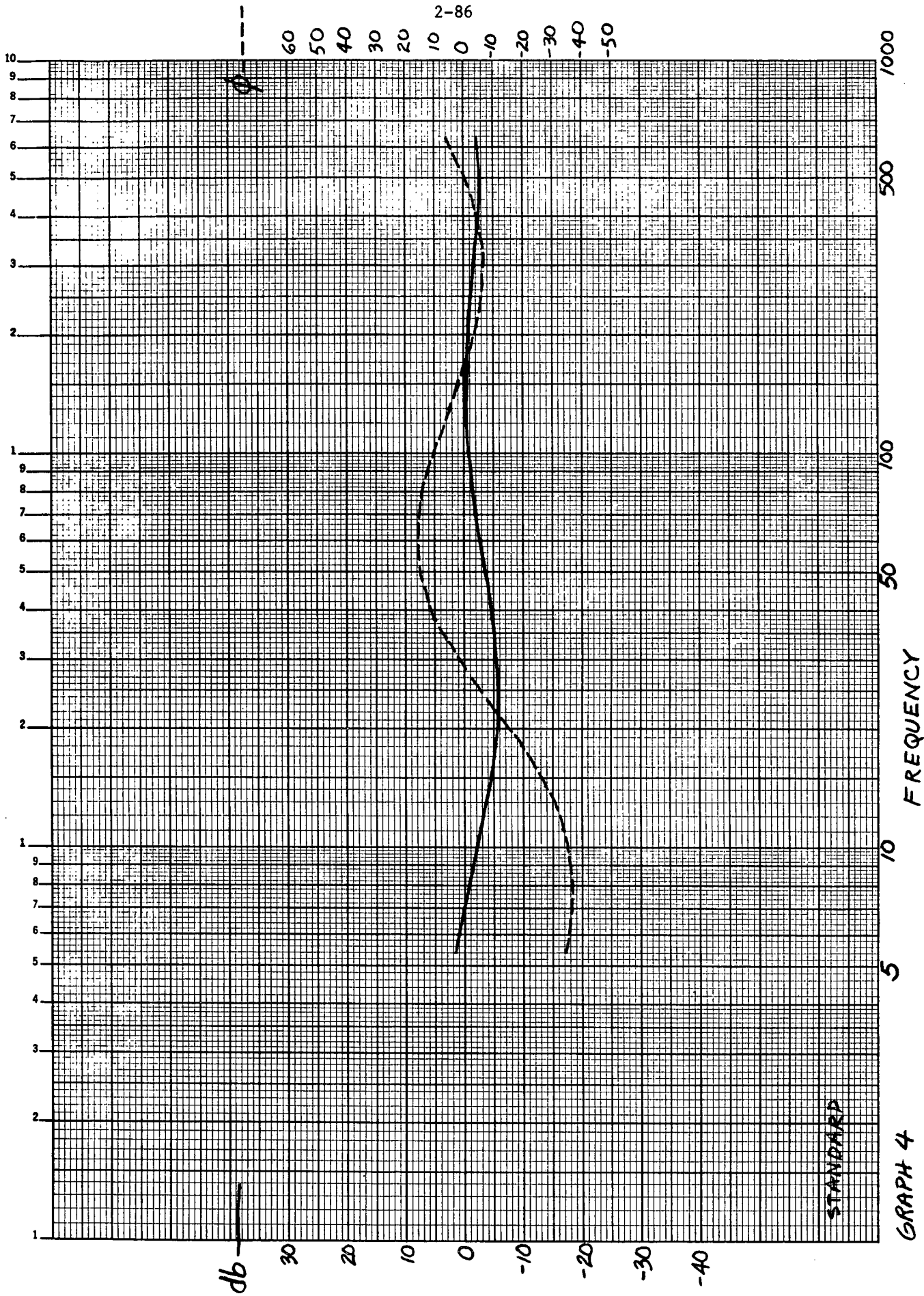
is limited to frequencies less than $1/2T$, or in this case, 500 Hz.

This condition is violated as is seen in the plot of the continuous

Butterworth design $G(s)$.

K·E SEMI-LOGARITHMIC 46 5813
 3 CYCLES X 140 DIVISIONS
 MADE IN U.S.A.
 KEUFFEL & ESSER CO.

2-86



GRAPH 4

The bilinear z-transform requires a prewarped frequency scale for the Butterworth $G(s)$ design, so

$$\omega_s = \frac{2}{T} \tan \left(\frac{\omega_z}{2} T \right).$$

	unwarped	warped
ω_u	200	200.67
ω_l	170	170.41
ω_n	600	618.67

The Butterworth design to be used in this case is

$$G(s) = \frac{1}{s^2 + \sqrt{2}s + 1} \quad \times \quad \frac{1}{s + 1} \quad \left| \quad \begin{array}{l} S = \frac{s(200.67 - 170.41)}{s^2 + (200.67)(170.41)} \\ s = s/618.67 \end{array} \right|$$

$$G(s) = 838.92 \frac{s^4 + 68392s^2 + 1.1694 \times 10^9}{s^4 + 1294.8s^3 + 69308s^2 + 4.4279 \times 10^7 s + 1.1694 \times 10^9} (s + 618.67)$$

$$G(s) = 838.92 \frac{(s^2 + 3.4199 \times 10^4)^2}{(s + 26.987)(s^2 + .70708s + 34199)(s + 1267.1)(s + 618.67)}$$

The bilinear z-transform is found by letting

$$D(z) = G(s) \left| \begin{array}{l} s = \frac{2}{T} \frac{1 - z^{-1}}{1 + z^{-1}} \end{array} \right.$$

$$D(z) = .19509 \frac{(1 - 1.9661z^{-1} + z^{-2})^2 (1 + z^{-1})}{(1 - .97337z^{-1})(1 - 1.9654z^{-1} + .99930z^{-2})(1 - .22433z^{-1})(1 - .52749z^{-1})}$$

The frequency response for this function is found with $z = e^{j\omega T}$ and is plotted in graph 5. Note that this plot closely matches graph 1.

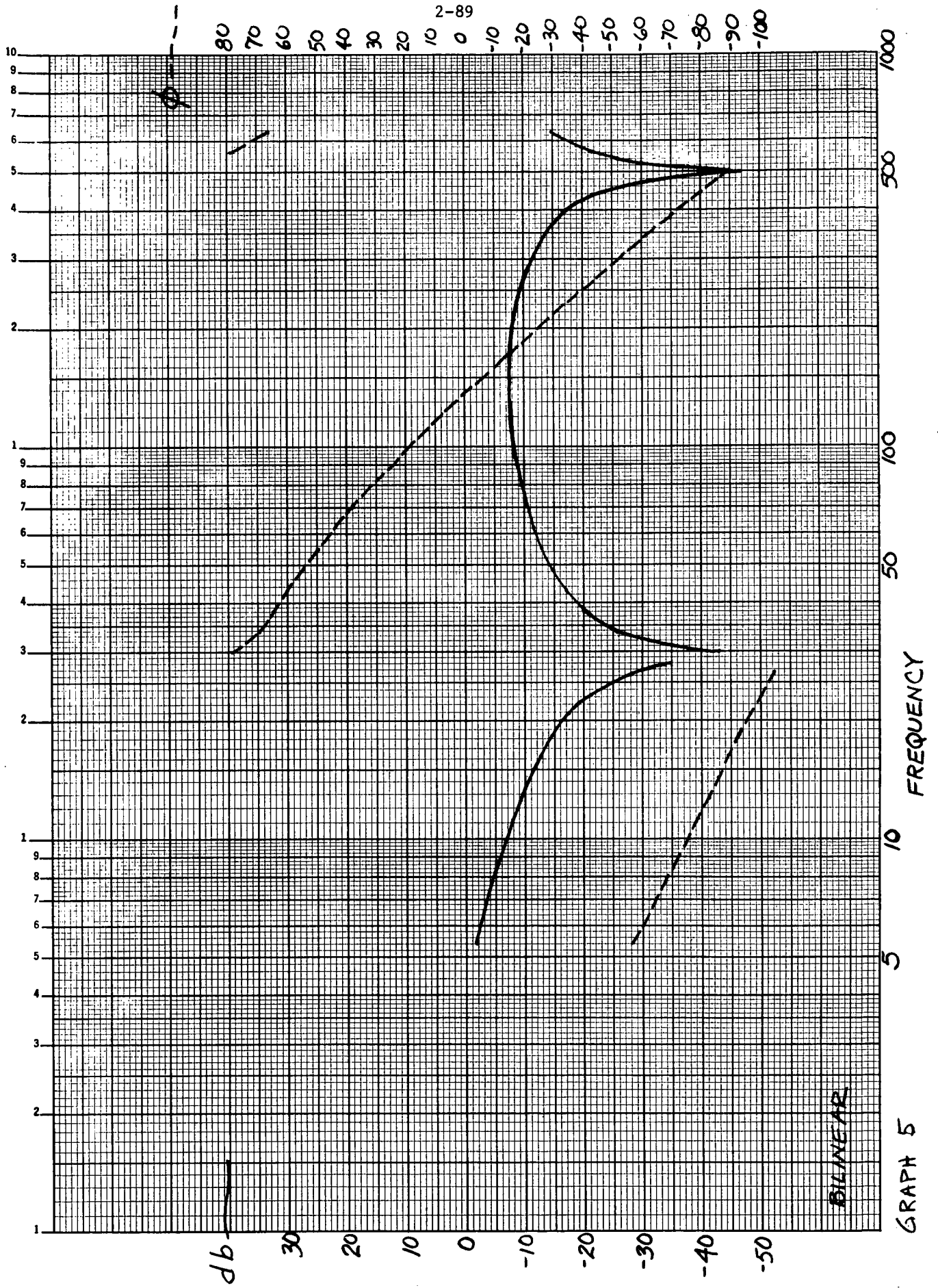
The Butterworth $G(s)$ may be factored as follows:

$$G(s) = 813.6 \frac{(s - j184.39)(s + j184.39)(s - j184.39)(s + j184.39)}{(s + 27.314)(s + .35375 + j184.39)(s + .35375 - j184.39)(s + 1244.8)(s + 600)}.$$

The matched z-transform is given by

$$D(z) = G(s) \left| \begin{array}{l} s + a = 1 - e^{-aT} z^{-1} \\ (s + u + iv)(s + u - iv) = 1 - 2e^{-uT} \cos vT z^{-1} + e^{-2uT} z^{-2} \end{array} \right.$$

K&E SEMI-LOGARITHMIC 46 5813
 3 CYCLES X 140 DIVISIONS
 KEUFFEL & ESSER CO.



and $D(1)$ is set equal to 1.356, the d.c. gain. Hence,

$$D(z) = .34607 \frac{(1-1.9661z^{-1}+z^{-2})^2}{(1-.97036z^{-1})(1-1.9654z^{-1}+.99929z^{-2})(1-.28800z^{-1})(1-.54881z^{-1})}$$

The frequency response of this function is plotted in graph 6. Note that the matched z-transform (like the bilinear) gives a good approximation to the response of graph 1.

Digital Compensators [38]

Digital filters are often employed as compensators for discrete control systems. Two common techniques for designing these compensators are root locus and Bode plots.

Root locus. A typical discrete-time closed-loop control system is demonstrated in Fig. 20a. Let

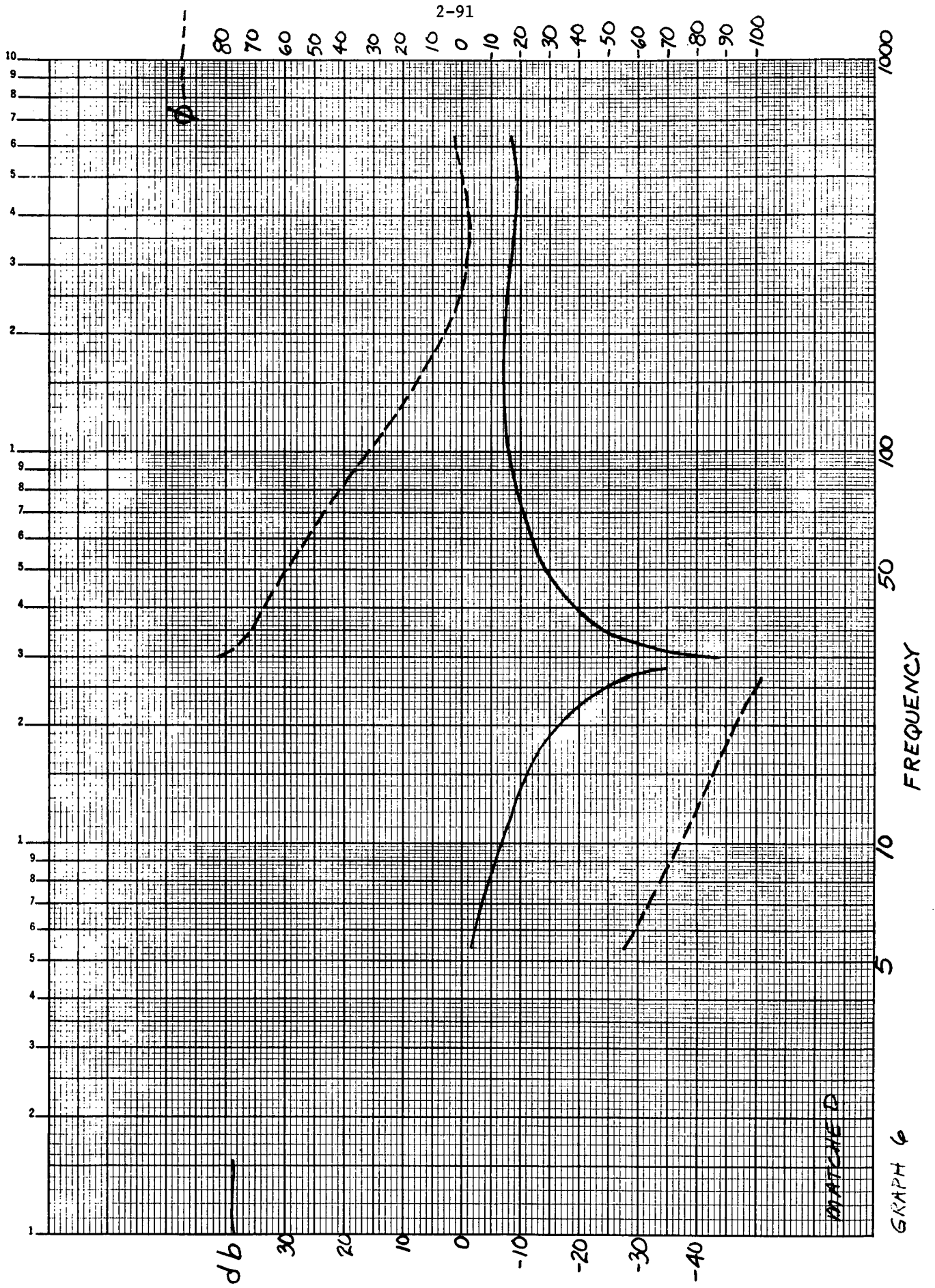
$$D(z) = K \frac{\sum_{i=0}^n a_i z^{-i}}{\sum_{i=0}^n b_i z^{-i}} \quad (2-23)$$

where K is a variable constant and $a_0 = b_0 = 1$. The root locus technique is outlined below

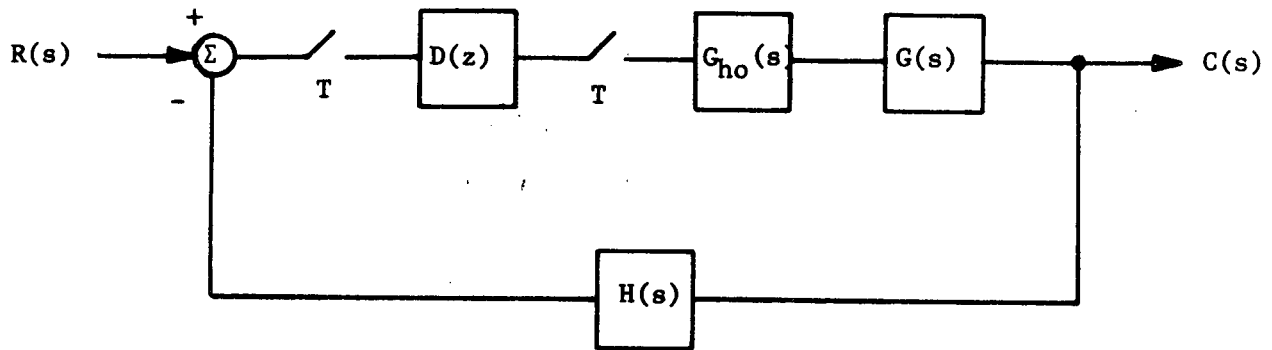
1. Find the characteristic equation

$$1 + D(z)Z[G_{ho}(s)G(s)H(s)]$$

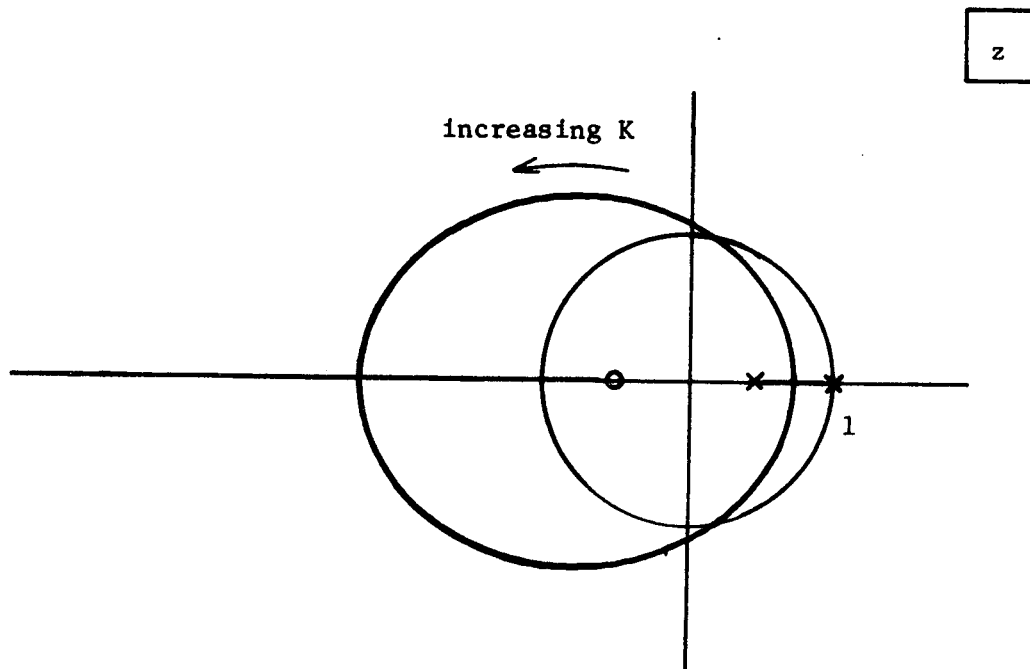
K&E SEMI-LOGARITHMIC 46 5813
 3 CYCLES X 140 DIVISIONS
 KEUFFEL & ESSER CO.



GRAPH 6



(a) Closed-loop Control System



(b) Typical Root Locus

Fig. 20. Root Locus

2. Place the poles and zeroes of $D(z)$ inside the unit circle in order to make the roots of the characteristic equation stable for some range of K .
3. Vary K from 0 to ∞ and solve for the closed-loop roots of the characteristic equation.
4. Choose an appropriate value for K .

In practice steps 2 and 3 are repeated on a trial and error basis. Once the procedure is complete, $D(z)$ in (2-23) is completely specified.

Bode plots. Bode plots are amplitude and phase plots for a transfer function constructed using the asymptotic behavior of simple first and second order factors in the numerator and denominator of the function $D(s)$. The plots are

$$db = 20 \log |D(j2\pi f)|$$

$$\phi = \angle D(j2\pi f) \quad .$$

Once the proper frequency response has been found, $D(s)$ may be mapped to the z -domain using the bilinear z -transform.

Frequency Sampling [3]

Earlier in this report the technique of implementing a finite duration impulse response filter in a recursive manner was presented. The coefficients must be integer powers of the first one for this technique to be applicable.

Nonlinear Programming [34]

Nonlinear programming can be used to design both recursive and nonrecursive digital filters. The filter is written as

$$H(z) = g \prod_{i=1}^S \frac{1 + a_i z^{-1} + b_i z^{-2}}{1 + c_i z^{-1} + d_i z^{-2}} \quad (2-24)$$

or

$$H(z) = g + \sum_{i=1}^S \frac{a_i + b_i z^{-1}}{1 + c_i z^{-1} + d_i z^{-2}} \quad (2-25)$$

An error function is formed

$$E_k = |H(e^{j2\pi f_k})|^2 - |F(2Wf_k)|^2 \quad k = 1, N \quad (2-26)$$

where f_k are the discrete frequencies, $2W$ is the sampling rate, and F is the desired continuous frequency response. Note that E_k is every where a differentiable function of a_i , b_i , c_i , d_i , and g . The errors E_k must satisfy

$$-L_k \leq E_k \leq U_k \quad k = 1, N. \quad (2-27)$$

From (2-27) we may define

$$\begin{aligned} G_k &\stackrel{\Delta}{=} QU_k - E_k \\ H_k &\stackrel{\Delta}{=} QL_k + E_k \end{aligned} \quad k = 1, N \quad (2-28)$$

where $Q > 0$.

A penalty function such as

$$Q + \sum_{k=1}^N \frac{r}{G_k} + \sum_{k=1}^N \frac{r}{G_k} \quad (2-29)$$

is formed. A suitable computer program (such as the Fletcher-Powell algorithm [39]) is used to minimize the penalty function with respect to Q , g , a_i , b_i , c_i , and d_i . Then the factor r is divided by a factor and the process is repeated until Q becomes nearly constant. If Q is less than unity the procedure stops; otherwise, increase the number of stages s of the filter and repeat the above procedure until a Q is found less than unity.

Optimal Digital Equivalent [40]

In this section the problem of determining an optimal digital equivalent $D(z)$ for a continuous filter $G(s)$ is considered (see Fig. 21). The coefficients of $D(z)$ are determined by fitting the input and outputs of the two filters. Let

$$e_{od}(j) = e_o(j), \quad (2-30)$$

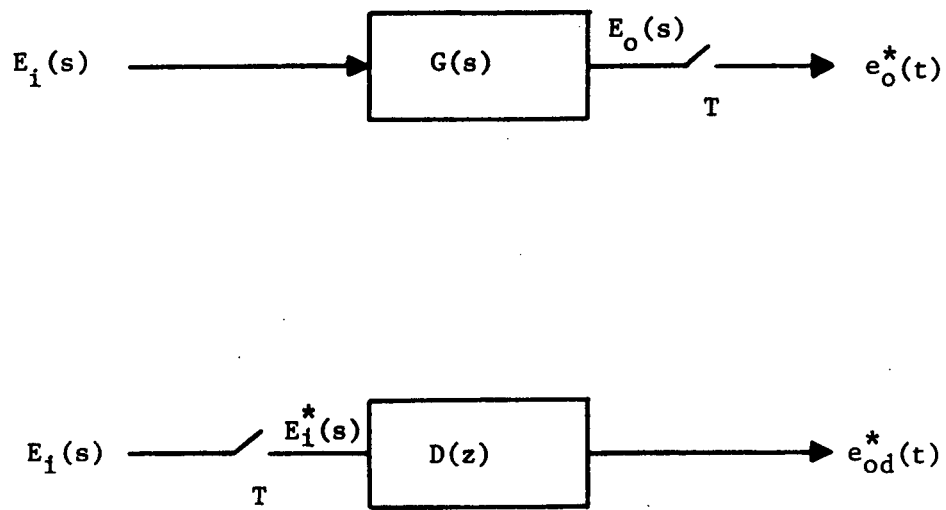


Fig. 21. The Equivalent Filters

and

$$D(z) = \frac{E_{od}(z)}{E_1(z)} = \frac{a_0 + a_1 z^{-1} + \cdots + a_{n-1} z^{-n+1}}{1 + b_1 z^{-1} + \cdots + b_n z^{-n}} \quad (2-31)$$

The problem then becomes one of choosing a_ℓ and b_ℓ in $D(z)$ such that (2-30) is satisfied. A difference equation for (2-31) is

$$e_{od}(j) - \sum_{\ell=0}^{n-1} a_\ell e_1(j-\ell) = \sum_{\ell=1}^n b_\ell e_{od}(j-\ell). \quad (2-32)$$

Substituting (2-30) into (2-32)

$$e(j) = \sum_{\ell=0}^{n-1} a_\ell e_1(j-\ell) - [e_o(j) + \sum_{\ell=1}^n b_\ell e_o(j-\ell)] \quad (2-33)$$

where $e(j)$ is driven to zero by minimizing $\overline{e^2(j)}$, the mean squared error.

In vector form (2-33) becomes $e(j) = \underline{q}^T(j) \underline{c} - e_o(j)$ where,

$$\underline{c}^T = [a_0, \cdots, a_{n-1}, -b_1, \cdots, -b_n] \quad (2-34)$$

$$\underline{q}^T(j) = [e_1(j), \cdots, e_1(j-n+1), e_o(j-1), \cdots, e_o(j-n)].$$

The mean square error is

$$\overline{e^2} = \lim_{N \rightarrow \infty} 1/(2N+1) \sum_{j=-N}^N e^2(j) \quad (2-35)$$

Equation (2-35) is minimized by

$$\frac{\partial e^2}{\partial \underline{c}} = 2 \lim_{N \rightarrow \infty} \frac{1}{(2N+1)} \sum_{j=-N}^N \underline{q}(j) e(j) = 0$$

or

$$\underbrace{\left(\lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{j=-N}^N \underline{q}(j) \underline{q}'(j) \right) \underline{c}}_R = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{j=-N}^N \underline{q}(j) e_o(j) \underbrace{\phantom{\lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{j=-N}^N}}_{\underline{r}}$$

and

$$\underline{c} = R^{-1} \underline{r} \tag{2-36}$$

Equation (2-36) gives the optimal filter coefficients. Let us examine R and r more closely:

$$R = \lim_{N \rightarrow \infty} \frac{1}{(2N+1)} \sum_{j=-N}^N \begin{bmatrix} e_1(j)e_1(j) \cdots e_1(j)e_1(j-n+1) & e_1(j)e_0(j-1) \cdots e_1(j)e_0(j-n) \\ \vdots & \vdots \\ e_1(j-n+1)e_1(j) \cdots e_1(j-n+1)e_1(j-n+1) & e_1(j-n+1)e_0(j-1) \cdots e_0(j-n+1)e_0(j-n) \end{bmatrix} \quad (2-37)$$

$$\begin{bmatrix} e_0(j-1)e_1(j) \cdots e_0(j-1)e_1(j-n+1) & e_0(j-1)e_0(j-1) \cdots e_0(j-1)e_0(j-n) \\ \vdots & \vdots \\ e_0(j-n)e_1(j) \cdots e_0(j-n)e_1(j-n+1) & e_0(j-n)e_0(j-1) \cdots e_0(j-n)e_0(j-n) \end{bmatrix}$$

2-99

$$r^T = \lim_{N \rightarrow \infty} \frac{1}{(2N+1)} \sum_{j=-N}^N \begin{bmatrix} e_1(j)e_0(j) \cdots e_1(j-n+1)e_0(j) \\ e_0(j-1)e_0(j) \cdots e_0(j-n)e_0(j) \end{bmatrix} \quad (2-38)$$

Equation (2-37) and (2-38) may be written

$$R = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \underline{r}^T = [E \ F]. \quad (2-39)$$

The elements of (2-39) are of the form

$$\begin{aligned} A: & \phi_{e_i e_i}(kT) \\ B: & \phi_{e_i e_o}(kT) \\ C: & \phi_{e_o e_i}(kT) \\ D: & \phi_{e_o e_o}(kT) \\ E: & \phi_{e_i e_o}(kT) \\ F: & \phi_{e_o e_o}(kT) \end{aligned} \quad (2-40)$$

where

$$\phi_{xy}(kT) = \lim_{N \rightarrow \infty} 1/(2N+1) \sum_{j=-N}^N x(j)y(j-k);$$

the ϕ_{xy} is the correlation function for discrete sequences. Since the input signal power spectrum $\phi_{e_i e_i}(s)$ and the analog filter $G(s)$ are known, (2-40) is determined by

$$\begin{aligned} \phi_{e_i e_i}(kT) &= \phi_{e_i e_i}(\tau) \Big|_{\tau = kT} = \mathcal{L}^{-1}[\phi_{e_i e_i}(s)] \Big|_{\tau = kT} \\ \phi_{e_i e_o}(kT) &= \mathcal{L}^{-1}[\phi_{e_i e_i}(s)G(s)] \Big|_{\tau = kT} = \phi_{e_o e_i}(kT), \end{aligned} \quad (2-41)$$

and

$$\Phi_{e_o e_o}(kT) = \left. {}^{-1} [\Phi_{e_1 e_1}(s) G(s) G(-s)] \right|_{\tau = kT}.$$

The digital filter determined above should have higher order than its analog counterpart so that the mean squared error will be small.

Sample Designs

In this section some example digital filters are listed.

Bandstop Filter

A digital bandstop filter was designed earlier in this chapter:

$$D(z) = \frac{.34607(1-1.9661z^{-1}+z^{-2})^2}{(1-.97036z^{-1})(1-1.9654z^{-1}+.99929z^{-2})(1-.2880z^{-1})(1-.54881z^{-1})} \quad (2-42)$$

The frequency response for $T = 0.001$ is shown in graph 6.

Digital Resonators

A digital oscillator is formed by placing complex poles on the unit circle:

$$D(z) = \frac{1}{1 - 2 \cos(2 fT)z^{-1} + z^{-2}} \quad (2-43)$$

where T is the sampling period and f is the frequency of oscillation.

Experimental results are available in [41].

Digital Differentiators [42]

The differentiator is a necessary part of many practical systems. The digital differentiator may take many forms; perhaps the best is a forth order recursive design shown below:

$$D(z) = A \frac{(1 - az^{-1})(1 - bz^{-1})(1 - cz^{-1})(1 - dz^{-1})}{(1 - ez^{-1})(1 - fz^{-1})(1 - gz^{-1})(1 - hz^{-1})} \quad (2-44)$$

where

$A = 0.36804011$	$e = -.10779165$
$a = 0.99999949$	$f = -.87602073$
$b = -0.86810806$	$g = 0.33494085$
$c = 0.32672838$	$h = 0.51312758$
$d = -.44183252$	

This differentiator was designed using nonlinear programming.

A nonrecursive wideband differentiator can be constructed for N samples by the relation

$$\begin{aligned} G_k &= k/(N/2) & k &= 0, N/2 \\ &= (N-k)/(N/2) & k &= N/2 + 1, N - 1. \end{aligned} \quad (2-45)$$

If the center samples are adjusted to optimally minimize the magnitude error for $N = 16$, then

$$G_{N/2} = 0.92890015$$

$$G_{(N/2)-1} = 0.86994255 \quad (2-46)$$

$$G_{(N/2)-2} = 0.75000000$$

yields a peak error magnitude of 7×10^{-5} for an 80% bandwidth.

Low-Pass Filters [43-45]

Reference [43] presents some 9 example nonrecursive low-pass filters of order 11. The designs are found using prolate spheroidal functions, least mean-square error, Fourier coefficients, windowing, binary weighting, and minimax techniques. The reader is referred to Table 1 of [43] for the appropriate coefficients.

III. COEFFICIENT QUANTIZATION

General [46]

One effect of finite wordlengths in digital computers is that the filter's parameters, or coefficients, must be chosen from a finite set of allowable values. Classical design procedures yield filter transfer functions with coefficients of arbitrary precision which must be altered for implementation using digital computing devices. One approach to this problem is to select a filter structure (programming form for the difference equations) which is not sensitive to coefficient inaccuracies. For example, realizing a filter directly allows a greater chance for instability than cascading or paralleling second order modules because it is well known that the roots of polynomials become more sensitive to parameter changes as the order of the polynomial increases.

Any programming form, or structure, produces a grid of allowable pole/zero locations in the z -plane. The proper structure to choose is one for which the grid is most dense in the areas at which the poles/zeros must be placed for a particular design. It is obvious that arbitrarily rounding or truncating denominator coefficients could cause poles to migrate outside the unit circle causing filter instability.

Instability Thresholds [47]

For low-pass filters, a measure of the number m_b of bits required to represent the coefficients of a stable filter may be expressed as

$$m_b = 1 - N \log_2(2\pi BT) \quad (3-1)$$

where

B = minimum attainable bandwidth

$$2^{i-1} \leq \binom{N}{[N/2]} < 2^i$$

for the direct programming form. For the cascade form

$$B \approx \frac{2 - (m_b - 2)/2}{2\pi T} \quad (3-2)$$

These stability thresholds are valid for filters designed using direct synthesis in the frequency domain for sine and tangent Butterworth low-pass filters. The results may be extended to other filter types.

Reduced Coefficient Wordlengths [48]

The cost of implementing a digital filter via a special-purpose computer is directly related to the wordlength of its coefficients. However, a short wordlength can cause large deviations in pole/zero placement. Hence a compromise must be found. The following procedure represents one solution to the problem.

Let the transfer function of the digital filter be

$$H(z) = \frac{\sum_{i=0}^m b_i z^{-i}}{\sum_{i=0}^m c_i z^{-i}} \quad (3-3)$$

where $c_o = 1$. If we examine the desired frequency response H_w around the unit circle

$$\begin{aligned} |H_w(e^{j\omega T})| &= 1 \quad \text{in passband} \\ &= 0 \quad \text{in stopband.} \end{aligned} \quad (3-4)$$

and is unspecified in the transition regions. If the maximum passband and stopband deviations are defined as δ_p and δ_s

$$|H_w(e^{j\omega T})| - |H(e^{j\omega T})| \leq \delta(e^{j\omega T})$$

or

$$\begin{aligned} \epsilon(e^{j\omega T}) &= \frac{1}{\delta_p} |1 - |H_n(e^{j\omega T})|| \quad \text{in PB} \\ &= \frac{1}{\delta_s} |H_n(e^{j\omega T})| \quad \text{in SB} \end{aligned} \quad (3-5)$$

where ϵ is the normalized error function and $H_n(z) = K H(z)$, a normalized transfer function.

The design of $H(z)$ minimizes $\max \epsilon$ such that

$$\max \epsilon < 1 \quad (3-6)$$

using standard minimax procedures. If (3-6) holds for a set of parameters \underline{a} , then justification for searching for a second set \underline{a}' of reduced word-length which also satisfies (3-6), where

$$\underline{a}^T = [b_0, \dots, b_m, c_0, \dots, c_m] \quad .$$

The coefficients are usually found for the cascade or parallel form.

The search for a new set \underline{a}' follows a modified univariate procedure which is described below:

1. Several sets of parameters, say 10, are stored in order of minimum max ϵ .
2. Perform a univariate search on the best set \underline{a}_1 . If no improvement can be found, try \underline{a}_2 .
3. Stop the procedure when no better improvement is found for any stored coefficient \underline{a}_i .

Generally, rounding of the coefficients is first performed. A univariate search reduces max ϵ by 25 to 50% over rounding, while a modified univariate search produces the best results reducing max ϵ by 25 to 50% over the univariate search.

In general, the development of synthesis procedures for quantized digital filter coefficients remains an active area of research.

IV. NONLINEARITIES IN FIXED POINT ARITHMETIC

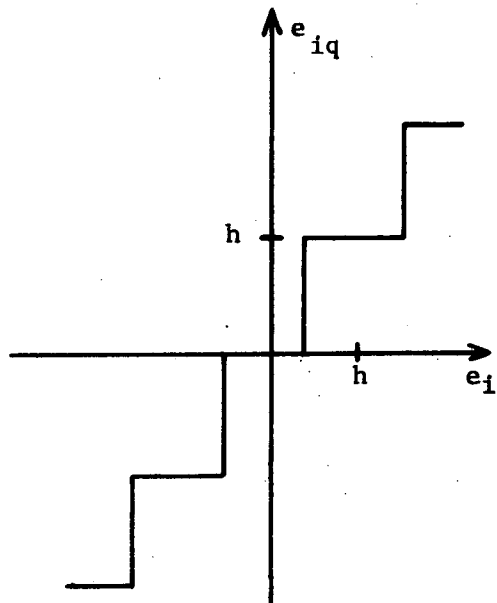
In digital computer implementations for digital filters, the restriction of finite wordlength produces several nonlinear phenomenon. Quantization occurs at the input sampler and in the internal arithmetic. Saturation and overflow also manifest themselves. Inaccuracies in coefficient representation has been discussed previously. Other notable effects which must be examined are limit cycles and deadbands.

Quantization Errors

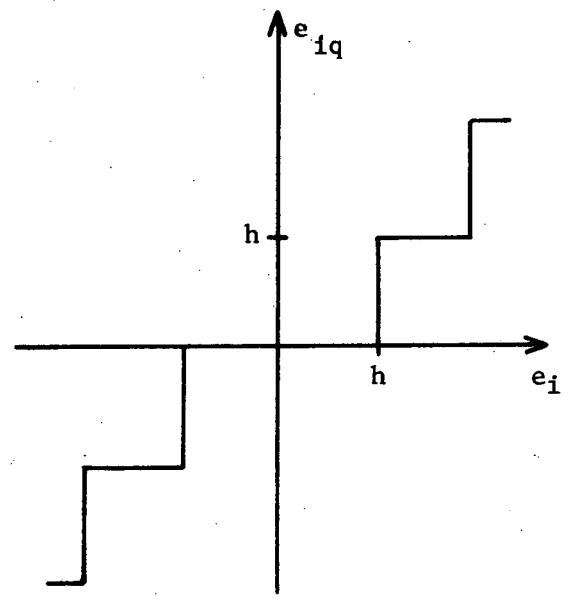
A digital filter specified by equation (3-3) is implemented by programming constant coefficient linear difference equations. The program for the difference equations will consist of the arithmetic operations, multiplication and addition (subtraction), and data transfer operations. The arithmetic unit of the computing device must be furnished binary numbers for the coefficients and variables of the difference equations. Since each coefficient and variable is represented by a finite number of binary digits, the binary numbers supplied to the arithmetic unit are quantized versions of the real numbers expected in the difference equation. Hence the digital filter introduces quantization errors into the system of which it is a part.

Quantizer Types [49]

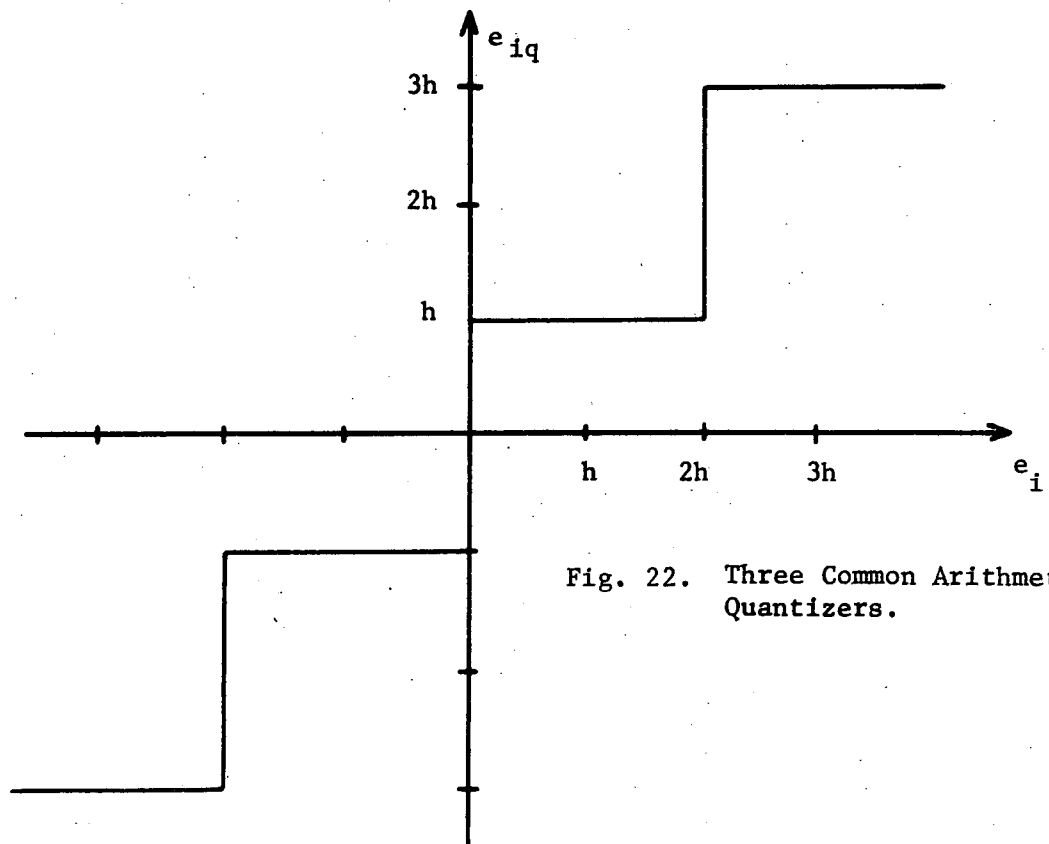
Signal amplitude quantization results from A/D conversion of the digital filter input signal, and from arithmetic operations within the computing device itself. Three common types of arithmetic quantizers are shown in Fig. 22; the step-length of each quantizer is h . Fig 22a illustrates the quantizing characteristic for a roundoff quantizer. The roundoff quantizer approximates the input signal e_i by the closest quantized value e_{iq} as follows:



(A) ROUND OFF



(B) TRUNCATION



(C) LSB-1

Fig. 22. Three Common Arithmetic Quantizers.

$$\begin{aligned}
-\frac{h}{2} &\leq e_i - e_{iq} < \frac{h}{2} \quad \text{for } e_i > 0 \\
-\frac{h}{2} &< e_i - e_{iq} \leq \frac{h}{2} \quad \text{for } e_i < 0.
\end{aligned}
\tag{4-1}$$

Therefore, the maximum error magnitude is $\frac{h}{2}$. The properties of the truncation quantizer is shown in Fig. 22b. This quantizer is less difficult to implement than the roundoff type; however, the approximation e_{iq} is less accurate:

$$\begin{aligned}
0 &\leq e_i - e_{iq} < h \quad \text{for } e_i > 0 \\
-h &< e_i - e_{iq} \leq 0 \quad \text{for } e_i < 0.
\end{aligned}
\tag{4-2}$$

Here the maximum error magnitude is h .

The third arithmetic quantizer presented in Fig. 22c is labeled LSB-1. In LSB-1 the least significant bit of quantized binary words is always set to "one." For this quantizer e_{iq} is never equal zero.

$$\begin{aligned}
-h &\leq e_i - e_{iq} < h \quad \text{for } e_i > 0 \\
-h &< e_i - e_{iq} \leq h \quad \text{for } e_i < 0.
\end{aligned}
\tag{4-3}$$

Again the maximum error magnitude is h .

Signal amplitude quantization at the A/D converter usually takes two forms. If the A/D converts the input signal magnitude to binary form, then the quantizer characteristic of Fig. 22b for truncation adequately

describes the effect of the A/D. However, if a bipolar A/D is used, the bipolar property is usually obtained by an offset bias voltage which causes the bipolar A/D quantization characteristic shown in Fig. 23. For this quantizer

$$0 \leq e_i - e_{iq} < h \quad (4-4)$$

and the maximum error is h .

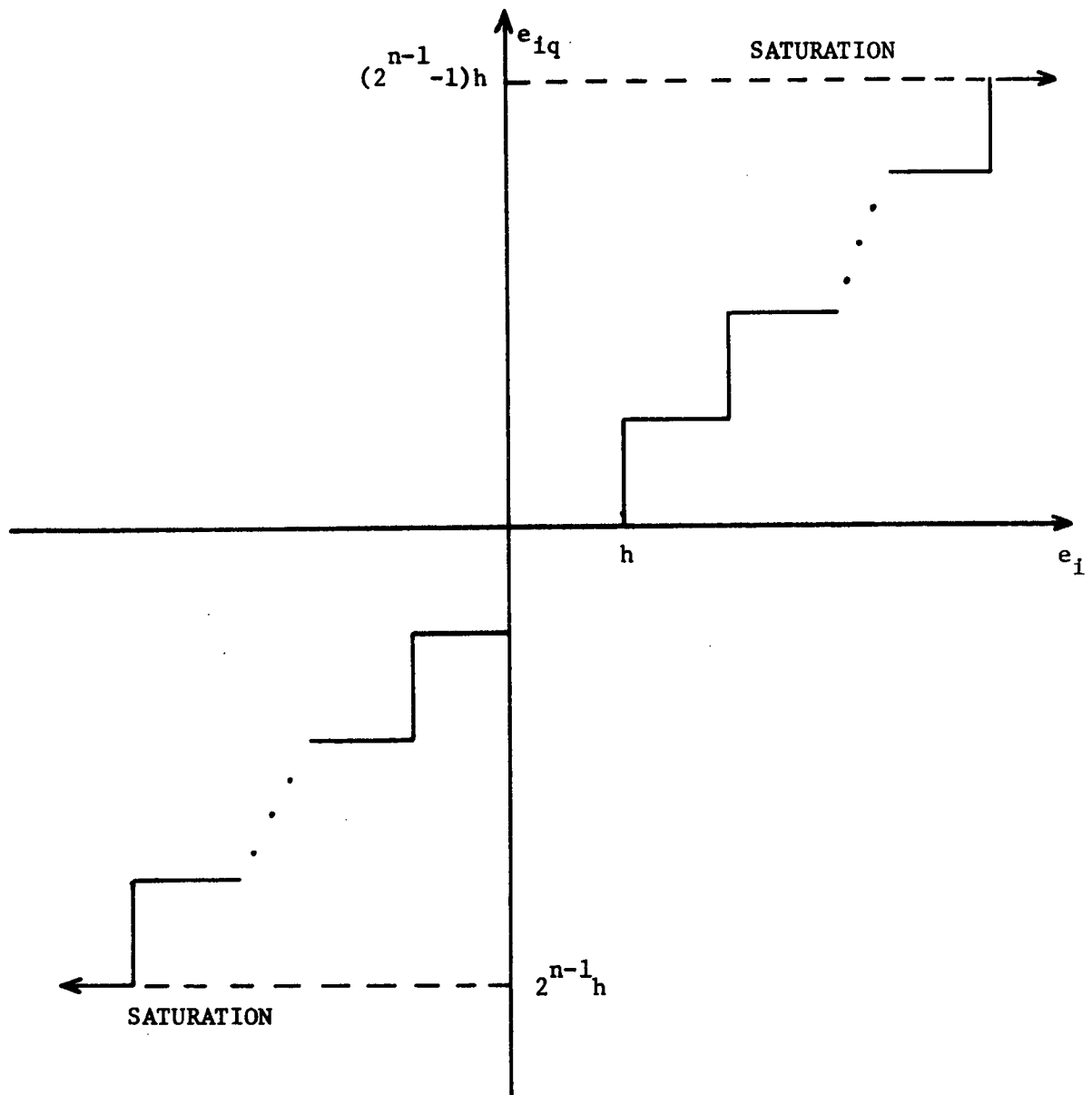
In summary, the maximum error magnitude introduced by a quantizer at a sampling instant is

$$\begin{aligned} \text{Roundoff: } h'_j &= h/2 \\ \text{Others: } h'_j &= h \end{aligned} \quad (4-5)$$

The quantizers of Figs. 22 and 23 may be represented in a system as an additional input error signal; this process is shown in Fig. 24. Using this model for the quantizers, their effect on system response will now be considered. Mathematical analysis of quantizing errors may generally be described as steady-state analysis, statistical analysis, and error bound analysis. Each of these analysis techniques will now be presented.

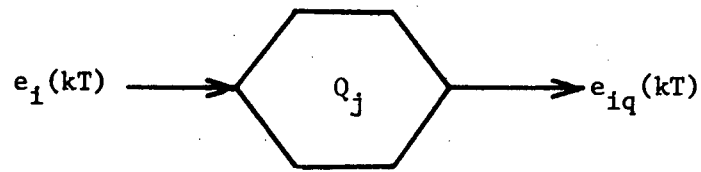
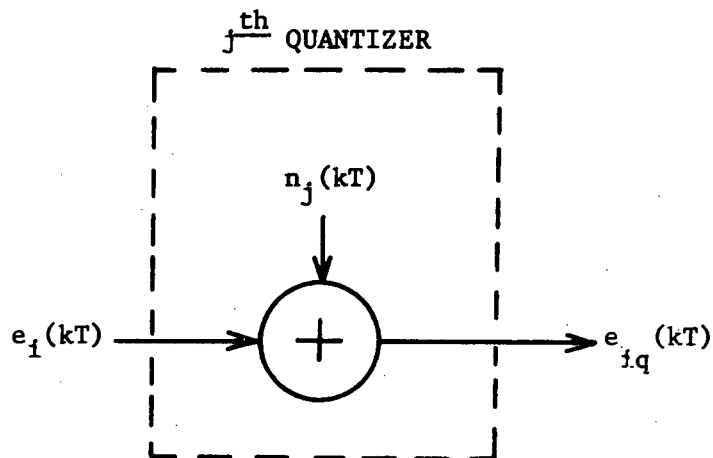
Steady-State Analysis [50]

The steady-state analysis may be divided into three steps. First, find the z -plane transfer functions $T_j(z)$ from the j th quantizing error



$N = \text{WORDLENGTH OF A/D}$

Fig. 23. Bipolar A/D.

(A) THE j^{th} QUANTIZER

(B) MATHEMATICAL REPRESENTATION

Fig. 24. Mathematical Model for a Quantizer.

source N_j to the system output, e_o . The total number of quantizers in the system is s . Hence,

$$E_{on}(z) = \sum_{j=1}^s T_j(z) N_j(z), \quad (4-6)$$

where $E_{on}(z)$ represents the output due to quantization errors.

Second, assume each error source is a step input of the maximum error amplitude h'_j for the type quantizer being analyzed. Therefore,

$$N_j(z) = \frac{h'_j}{1 - z^{-1}} \quad (4-7)$$

Substituting (4-7) into (4-6)

$$E_{on}(z) = \sum_{j=1}^s \frac{T_j(z) h'_j}{1 - z^{-1}}. \quad (4-8)$$

Lastly, apply the z-transform final value theorem [$y(\infty) = \lim_{z \rightarrow 1} (1 - z^{-1}) \cdot Y(z)$] to (4-8); thus,

$$\begin{aligned} e_{on}(\infty) &= \lim_{z \rightarrow 1} \sum_{j=1}^s T_j(z) h'_j \\ &= \sum_{j=1}^s \left[\lim_{z \rightarrow 1} T_j(z) \right] h'_j. \end{aligned}$$

If one defines

$$K_{ssj} = \lim_{z \rightarrow 1} T_j(z) \quad (4-9)$$

then

$$e_{on}(\infty) = \sum_{j=1}^S K_{ssj} h'_j \quad (4-10)$$

Equation (4-10) may be used to evaluate the effect of each quantizer on the system output under steady-state conditions.

Another technique for finding the K_{ssj} weighting constants for (4-10) is derived as follows. The standard z-transform for $t_j(t)$ is

$$T_j(z) = \sum_{k=0}^{\infty} t_j(kT) z^{-k} \quad (4-11)$$

where kT represents a sample instant. Hence (4-9) becomes

$$K_{ssj} = \sum_{k=0}^{\infty} t_j(kT)$$

If $t_j(kT)$ tends to zero as kT gets large, say NT ,

$$K_{ssj} \approx \sum_{k=0}^N t_j(kT) \quad (4-12)$$

may be used in (4-10) to calculate the steady-state error. The terms $t_j(kT)$ in (4-12) may be obtained from a simulation of the system by applying $N_j(z) = 1$, a discrete impulse function. Note that the weighting constants are functions of the system characteristics and not of the quantizers.

Statistical Analysis [51]

If the input signal to a roundoff quantizer Q_j has a dynamic range of more than three step intervals h_j , the effect of the quantizer may be determined by replacing it with a unity gain and an additive white noise $n_j(kT)$ (see Fig. 24) with a rectangular amplitude distribution density function $p(n_j)$ of bounds $\pm h_j/2$ and height $1/h_j$. The LSB-1 quantizer can also be replaced in this manner with $p(n_j)$ bounded by $\pm h_j$ and $1/2h_j$. The truncation quantizer cannot be represented exactly in this manner, but this technique does give a good approximation with $p(n_j)$ bounded by $\pm h_j$ and $1/2h_j$. Let us continue by analyzing the roundoff case which can be easily extended to the others.

The variance $\sigma_{n_j}^2$ of this rectangular distribution is

$$\sigma_{n_j}^2 = \int_{-\infty}^{\infty} n_j^2 p(n_j) dn_j = \frac{h_j^2}{12} \quad (4-13)$$

When the dynamic range of the input signal is greater than three quantization levels, the noise input of the quantizer is essentially uncorrelated between successive sampling intervals, and the autocorrelation of the quantization noise becomes

$$\begin{aligned} \phi_{n_j n_j}(\tau) &= \sum_{n=-\infty}^{\infty} \sigma_{n_j}^2 (T - |\tau|)/T & |\tau| \leq T \\ &= 0 & |\tau| > T \end{aligned} \quad (4-14)$$

The sampled power density spectrum is defined by

$$\phi_{n_j n_j}(z) = \sum_{n=-\infty}^{\infty} \phi_{n_j n_j}(nT) z^{-n} = \sigma_{n_j}^2 = \frac{h_j^2}{12} \quad (4-15)$$

The mean-squared error output due to one quantizer error is

$$e_{on}^2(kT) = 1/2\pi i \int_{\Gamma} \phi_{n_j n_j}(z) T_j(z) T_j(z^{-1}) dz/z \quad (4-16)$$

where $T_j(z) = E_{on}(z)/N_j(z)$, Γ is the unit circle, and $i = \sqrt{-1}$. Substituting (4-15) into (4-16) and assuming that the total rms output error is bounded by the sum of the s rms errors due to the quantizer inputs yields

$$[e_{on}]_{rms} \leq \sum_{j=1}^s K_{stj} h_j \quad (4-17)$$

where

$$K_{stj} = \left[\frac{1}{24\pi i} \int_{\Gamma} T_j(z) T_j(1/z) dz/z \right]^{1/2}. \quad (4-18)$$

The integral in (4-18) may be evaluated by calculating the residues of the integrand.

Another technique for calculating the mean-square output error is by using the following identity:

$$\frac{1}{2\pi i} \int_{\Gamma} F(z) F(1/z) dz/z = \sum_{k=0}^{\infty} f(kT)^2$$

Hence (4-18) becomes

$$K_{stj} = \left[\frac{1}{12} \sum_{k=0}^{\infty} t_j(kT)^2 \right]^{1/2}$$

where $t_j(kT)$ is found from the impulse response in a simulation of the system. If $t_j(kT)$ converges to zero for k large, say N , then

$$K_{stj} \approx \left[\frac{1}{12} \sum_{k=0}^N t_j(kT)^2 \right]^{1/2} \quad (4-19)$$

This relation may be used instead of (4-18) for many applications.

Equations (4-18) and (4-19) are for roundoff only. They should be altered by substituting $h_j = 2h'_j$ into (4-17) for the general case.

Quantization Error Bounds [52]

Consider an n^{th} order system described by

$$\begin{aligned} \underline{x}(k+1) &= A\underline{x}(k) + D\underline{r}(k) \\ e_o(k) &= \underline{c}^T \underline{x}(k) + \underline{d}^T \underline{r}(k) \end{aligned} \quad (4-20)$$

where $\underline{r}(k)$ is a vector of the system inputs and $e_o(k)$ is the output.

The sampling interval T has been eliminated for convenience. The introduction of quantizers into the system results in

$$\begin{aligned} \underline{x}^q(k+1) &= A\underline{x}^q(k) + D\underline{r}(k) - B\underline{q}(k) \\ e_o^q(k) &= \underline{c}^T \underline{x}^q(k) + \underline{d}^T \underline{r}(k) - \underline{f}^T \underline{q}(k) \end{aligned} \quad (4-21)$$

where $\underline{q}(k)$ represents a vector of the s quantizer error inputs $n_j(k)$. A state variable representation for the quantization error

$$\underline{v}(k) = \underline{x}(k) - \underline{x}^q(k)$$

$$e_{on}(k) = e_o(k) - e_o^q(k) \quad (4-22)$$

results from subtracting (4-21) from (4-20)

$$\underline{v}(k+1) = A\underline{v}(k) + B\underline{q}(k)$$

$$e_{on}(k) = \underline{c}^T \underline{v}(k) + \underline{f}^T \underline{q}(k). \quad (4-23)$$

The general solution for (4-23) is

$$\underline{v}(N) = A^N \underline{v}(0) + \sum_{\ell=0}^{N-1} A^\ell B \underline{q}(N-1-\ell) \quad (4-24)$$

$$e_{on}(N) = \underline{c}^T A^N \underline{v}(0) + \sum_{\ell=0}^{N-1} \underline{c}^T A^\ell B \underline{q}(N-1-\ell) + \underline{f}^T \underline{q}(N).$$

For N large,

$$\begin{aligned} \underline{v}(N) &= \sum_{j=1}^s \left(\sum_{\ell=0}^{N-1} A^\ell \underline{b}_j \right) q_j(N-1-\ell) \\ e_{on}(N) &= \sum_{j=1}^s \left(\sum_{\ell=0}^{N-1} \underline{c}^T A^\ell \underline{b}_j \right) q_j(N-1-\ell) + \underline{f}^T \underline{q}(N). \end{aligned} \quad (4-25)$$

where $q_j(N)$ is the j^{th} quantizer and \underline{b}_j is the j^{th} column of the $n \times s$ matrix B. Since, if $a = bc + de$, $|a| \leq |b||c| + |d||e|$

$$|\underline{v}(N)| \leq \sum_{j=1}^s \left(\sum_{\ell=0}^{N-1} |A^{\ell} \underline{b}_j| \right) |q_j(N-1-\ell)|$$

and

$$|\underline{v}(N)|_{\max} \leq \sum_{j=1}^s \left(\sum_{\ell=0}^{N-1} |A^{\ell} \underline{b}_j| \right) h'_j \quad (4-26)$$

where $h'_j = |q_j(N-1-\ell)|_{\max}$ is given in (4-5). Similarly,

$$|e_{on}(N)|_{\max} \leq \sum_{j=1}^s \left(\sum_{\ell=0}^{N-1} |\underline{c}^T A^{\ell} \underline{b}_j| \right) h'_j + \sum_{j=1}^s |f_j| h'_j. \quad (4-27)$$

In another form,

$$|\underline{v}(N)|_{\max} \leq \sum_{j=1}^s \underline{m}_j h'_j \quad (4-28)$$

$$|e_{on}(N)|_{\max} \leq \sum_{j=1}^s K_{ubj} h'_j$$

where

$$\underline{m}_j = \sum_{\ell=0}^{N-1} |A^{\ell} \underline{b}_j|, \quad j = 1, s$$

$$K_{ubj} = \sum_{\ell=0}^{N-1} \left(|\underline{c}^T A^{\ell} \underline{b}_j| \right) + |f_j|, \quad j = 1, s. \quad (4-29)$$

Note that (4-29) gives weighting vectors \underline{m}_j and weighting constants K_{ubj} which are functions of the system and not of the quantizers. Hence, (4-28)

and (4-29) are useful in helping to choose quantization error schemes for systems with digital filters.

A second method for bounding the output error due to quantizer Q_j is from the transfer function

$$T_j(z) = \frac{E_{on}(z)}{N_j(z)}.$$

The impulse response is found with $N_j(z) = h_j'$. Therefore,

$$E_{on}(z) = T_j(z)h_j' = \left[\sum_{k=0}^{\infty} t_j(k)z^{-k} \right] h_j'. \quad (4-30)$$

To calculate the worst case output error e_{on} due to quantizer Q_j

$$|e_{on}(N)|_{\max} \leq \left[\sum_{k=0}^{\infty} |t_j(k)| \right] h_j'. \quad (4-31)$$

Similar to the argument employed for equations (4-12) and (4-19)

$$|e_{on}(N)|_{\max} \leq \sum_{j=1}^s K_{ubj} h_j'$$

where

$$K_{ubj} \approx \sum_{k=0}^N |t_j(k)|, \quad j = 1, s. \quad (4-32)$$

Equation (4-32) may be used to calculate the weighting constants K_{ubj} instead of (4-29).

A summary of the results of the quantization analysis presented in this section is displayed in Table 1.

TABLE 1: Quantization Analysis

Figure of Merit = $\sum_{j=1}^S \text{Constant}_j h_j'$		
Analysis Method	Figure of Merit	Constant _j
Steady State	Steady State Error	$K_{ssj} = \lim_{z \rightarrow 1} T_j(z)$ $K_{ssj} \approx \sum_{k=0}^N t_j(kT)$
Statistical	Root mean square error	$K_{stj} = \left[\frac{1}{6\pi i} \int_{\Gamma} T_j(z) T_j(1/z) \frac{dz}{z} \right]^{1/2}$ $K_{stj} \approx \left[\frac{1}{3} \sum_{k=0}^N t_j(kT)^2 \right]^{1/2}$
Error Bound	Maximum error	$K_{ubj} \approx \sum_{\ell=0}^{N-1} \underline{c}^T A^{\ell} \underline{b}_j + f_j $ $K_{ubj} \approx \sum_{k=0}^N t_j(kT) $

Open-Loop vs. Closed-Loop [53]

The quantization analysis procedures above are equally applicable to open-loop or closed-loop systems. However, open-loop analysis of the digital filter itself is perhaps the easier approach. It has been shown in [49,53] that open-loop analysis can give satisfactory results even if the filter is to reside in a closed-loop system.

Limit Cycles and Deadband Effects [46,54,55]

Consider the digital filter

$$y_n = x_n + \beta y_{n-1} \quad , \quad \beta = 0.5 \quad (4-33)$$

implemented in fixed-point arithmetic with roundoff quantization. If the input x_n is a impulse function of value $7/8$

$$\begin{aligned} y_0 &= 7/8 \\ y_1 &= 1/2 \\ y_2 &= 1/4 \\ y_3 &= 1/8 \\ y_n &= 1/8 \quad n \geq 4 \end{aligned} \quad (4-34)$$

is the resulting output sequence. Ideally the output should go to zero. This type error is called a limit cycle, and the amplitude intervals within limit cycles are called deadbands. The deadband for (4-33) is

$$|y_{n-1}| - \beta |y_{n-1}| \leq \left(\frac{1}{2}\right) 2^{-b}$$

where b is the number of magnitude bits. Hence

$$|y_{n-1}| \leq \frac{\left(\frac{1}{2}\right) 2^{-b}}{1 - |\beta|} \quad (4-35)$$

For the second-order filter

$$y_n = x_n - \beta_1 y_{n-1} - \beta_2 y_{n-2} \quad (4-36)$$

the deadband is

$$|y_{n-2}| \leq \frac{2^{-b-1}}{1 - |\beta_2|} \quad (4-37)$$

The deadband for higher order filters is directly dependent upon the programming form. In general, the parallel form yields better results because one need not be concerned with the ordering of cascaded sections [54].

Saturation and Overflow [56]

When a filter is implemented in one's or two's complement arithmetic and signal values exceed the finite register length upper limit, a overflow condition occurs and the results usually changes sign. This condition can cause large limit cycles, called overflow oscillations, to be excited. These oscillations may be avoided by using saturation arithmetic as designed in [57,79]. One must be wary of this solution for in many closed-loop control systems, saturating the signals causes system instability. Saturation changes the filter output which effectively alters, temporarily, the transfer function.

Dynamic Range [46]

The dynamic range of a binary signal x_n of $b + 1$ bits is

$$0 \leq |x_n| \leq 2^b - 1. \quad (4-38)$$

Increasing the number of bits by one doubles the dynamic range. As seen in the last section, it is important that the dynamic range of a digital filter in many applications never be exceeded. Hence, several techniques may to employed to find b .

One technique finds the least upper bound on the signal x_n and uses (4-38) to specify b and hence this limit can never be exceeded. More practical solutions use simulation of the filter with typical inputs to define the dynamic range of the internal variables. Sometimes statistical methods are used for non-deterministic input signals.

V. NONLINEARITIES IN FLOATING POINT ARITHMETIC

In the past there has been little emphasis placed on research and analysis of quantization errors at the output of a floating point filter, the reason for this being that most filter implementations use fixed point arithmetic. Sandberg [58] was the first to study quantization error analysis for floating point filters with [59-62] being more recent.

As in the case of fixed point filters, quantization error for floating point filters has three sources due to finite word length. They are

- 1) the quantization of the input signal x_n into a set of discrete levels;
- 2) the representation of the coefficients of the filter, a_k and b_k , by a finite number of bits;
- 3) the accumulation of roundoff errors caused by arithmetic operations.

Notation

If we assume the ideal output of the filter is w_n and the actual output y_n , the error at the output of the n^{th} sample e_n may be defined as

$$e_n = y_n - w_n \quad (5-1a)$$

where

$$w_n = \sum_{k=0}^M a_k x_{n-k} - \sum_{k=1}^N b_k w_{n-k} \quad (5-1b)$$

Before the effects of the above error sources are discussed, the representation of floating point numbers with a fixed number of bits should be considered.

A floating point number is written in the form $(\text{sgn})2^b \cdot a$, where b is a binary integer called the exponent and a is a fraction between $1/2$ and 1 called the mantissa. As expected, the range of numbers that can be represented is determined by the number of bits of the exponent. In order to represent a number v in floating point form with a t -bit mantissa, the smallest integer exceeding $\log_2 v$ is first determined. This number is denoted by $\lceil \log_2 v \rceil$. The binary expansion of the fraction $v/\lceil \log_2 v \rceil$ is then rounded to t bits. If $(v)_t$ denotes the t -bit mantissa floating point approximation, it is seen that

$$(v)_t = v(1 + \epsilon) \quad (5-2)$$

where the error is bounded by $-2^{-t} \leq \epsilon < 2^{-t}$, or $[-2, 2)$.

Error Sources

Both addition and multiplication in floating point arithmetic introduce roundoff error. Let $(v_1 \cdot v_2)_t$ and $(v_1 + v_2)_t$ denote, respectively, the actual computed product and sum of two numbers v_1 and v_2 ; then

$$(v_1 \cdot v_2)_t = (v_1 \cdot v_2)(1 + \delta) \quad (5-3)$$

$$(v_1 + v_2)_t = (v_1 + v_2)(1 + \epsilon) \quad (5-4)$$

where the errors δ and ϵ are bounded by $[-2^{-t}, 2^{-t}]$.

The above errors will be regarded as random quantities and they will be uniformly distributed in their range $[-2^{-t}, 2^{-t}]$. Making these and the above assumptions, a statistical approach will be discussed which predicts floating point quantization errors.

First, consider the effect of input quantization. Supposing the quantizer has equal step size h , the input to the filter is $x_n + e_n^Q$ where each e_n^Q is bounded by $-(h/2) \leq e_n^Q < (h/2)$. Since the filter is linear, the output is the sum of the two components, x_n and e_n^Q . In determining the effect of input quantization, e_n^Q is considered as white noise with a zero mean and variance $h^2/12$. The steady-state output component due to e_n^Q is a zero-mean wide-sense-stationary (w.s.s.) sequence with power spectral density

$$H(z)H(1/z)(h^2/12) \quad (5-5)$$

where $H(z)$ is the transfer function of the filter as repeated below

$$H(z) = \left(\sum_{k=0}^M a_k z^{-k} \right) / \left(1 + \sum_{k=1}^N b_k z^{-k} \right) \quad (5-6)$$

The effect of coefficient inaccuracy on roundoff accumulation has been ignored.

An expression for the mean-squared value of the error at the filter's output due to input quantization is obtained by integrating the power spectral density (Equation (5-5)). It is equal to

$$1/2\pi j \oint [H(z)H(1/z) (h^2/12)]/z dz \quad (5-7)$$

Coefficient Quantization

Considering the effect of coefficient quantization, it is seen that each coefficient is replaced by its t -bit representation according to (5-2). This means the coefficient a_k is replaced by $(a_k)_t$, which equals $a_k(1 + \alpha_k)$, with α_k bounded in absolute value by 2^{-t} . Likewise, each b_k is replaced by $(b_k)_t$ which is $b_k(1 + \beta_k)$. Because of this, it is obvious that the filter characteristics will change. The problem can be approached in several ways. The first, and the simplest, is to compute the frequency response of the actual filter with t -bit rounded coefficients by using the actual transfer function

$$[H(z)]_t = \left(\sum_{k=0}^M (a_k)_t z^{-k} \right) / \left(1 + \sum_{k=1}^N (b_k)_t z^{-k} \right) \quad (5-8)$$

and then comparing the result with the ideal response of the original design.

Coefficient rounding can cause movements of the poles and zeroes of the transfer function. When this happens, network sensitivity theory can be applied to study the changes of the filter response. If the poles of $H(z)$ are at z_i , $i = 1, N$, and the poles of $[H(z)]_t$ are at $z_i + \Delta z_i$, it can be shown that

$$\Delta z_i = \sum_{k=1}^N [(z_i^{k+1}) / \prod_{\substack{m=1 \\ m \neq i}}^N (1 - (z_i/z_m))] / \Delta a_k \quad (5-9)$$

where Δa_k is the change in the coefficient a_k . Likewise, results can be obtained for the movement of the zeroes. The change in the filter response can be studied from these movements.

Instability of a filter may occur, due to coefficient error, when a filter has poles that are close to the unit circle in the z -plane. The problem can be serious when the sampling rate of the filter is relatively high, even for low order filters in the direct form. Kaiser [62] has demonstrated that for an N^{th} -order low-pass filter operating at a sampling rate of $1/T$ with distinct poles at $e^{-p_k T}$, stability is guaranteed if the number of bits used m_b satisfies the inequality

$$m_b > [-\log_2 [5\sqrt{N} / 2^{N+2}) (\prod_{k=1}^N p_k T)]] \quad (5-10)$$

where the bracket denotes the smallest integer exceeding the quantity inside. It is also possible to extend the result to include multiple

C

poles and to derive similar results for filters of other than low-pass type.

The effect of coefficient inaccuracy is more pronounced for a high-order filter when it is realized in the direct form than when it is realized in the parallel or cascade form, which suggests the parallel or cascade form should be used for high-order filters when possible. Further details on coefficient quantization are given in Chapter 3.

Output Error

Roundoff accumulation error for floating point filters [59-61] is quite different from that of fixed point filters and consequently will be treated with more depth than that of fixed point. The errors introduced are relative to Equations (5-2), (5-3) and (5-4). The calculation of the statistical mean-squared error at the output will be discussed for the direct programming form with the understanding that extension to other forms is easily accomplished [61].

It has been shown that for floating point arithmetic the actual filter coefficients are $a_k(1 + \alpha_k)$ and $b_k(1 + \beta_k)$ where α_k and β_k are bounded in absolute value by 2^{-t} . The actual computed output y_n is given by

$$y_n = f\ell \left[\sum_{k=0}^M a_k(1 + \alpha_k)x_{n-k} + \sum_{k=1}^N b_k(1 + \beta_k)y_{n-k} \right] \quad (5-11)$$

where $fl[\]$ denotes the actual computed result by floating point arithmetic of the quantity inside the brackets. It is assumed that the computation of (5-11) is carried out in the following order: the products $a_k(1 + \alpha_k)x_{n-k}$ and $b_k(1 + \beta_k)y_{n-k}$ are first formed; the two sums are then calculated; and finally the difference is taken to give y_n . Each of these arithmetic operations introduces a round-off error which is characterized by (5-3) and (5-4). A flowgraph of this operation may be drawn, as is shown in Fig. 25, which includes all the roundoff error introduced in the calculation of y_n . From Fig. 25, it is seen that $\delta_{n,k}$ is introduced when the product of $a_k(1 + \alpha_k)x_{n-k}$ is formed, and $\delta_{n,1}$ is introduced when the computed products of $a_0(1 + \alpha_0)x_n$ and $a_1(1 + \alpha_1)x_{n-1}$ are added. The actual output y_n is then

$$y_n = \sum_{k=0}^M a_k(1 + \alpha_k)\theta_{n,k}x_{n-k} - \sum_{k=1}^N b_k(1 + \beta_k)\phi_{n,k}y_{n-k} \quad (5-12)$$

where

$$\begin{aligned} \theta_{n,0} &= (1 + \xi_n)(1 + \delta_{n,0}) \prod_{i=1}^M (1 + \zeta_{n,i}) \\ \theta_{n,k} &= (1 + \xi_n)(1 + \delta_{n,k}) \prod_{i=k}^M (1 + \zeta_{n,i}), \quad k = 1, 2, \dots, M \\ \phi_{n,1} &= (1 + \xi_n)(1 + \epsilon_{n,1}) \prod_{i=2}^N (1 + \eta_{n,i}) \\ \phi_{n,k} &= (1 + \xi_n)(1 + \epsilon_{n,k}) \prod_{i=k}^N (1 + \eta_{n,i}), \quad k = 2, 3, \dots, L \end{aligned} \quad (5-13)$$

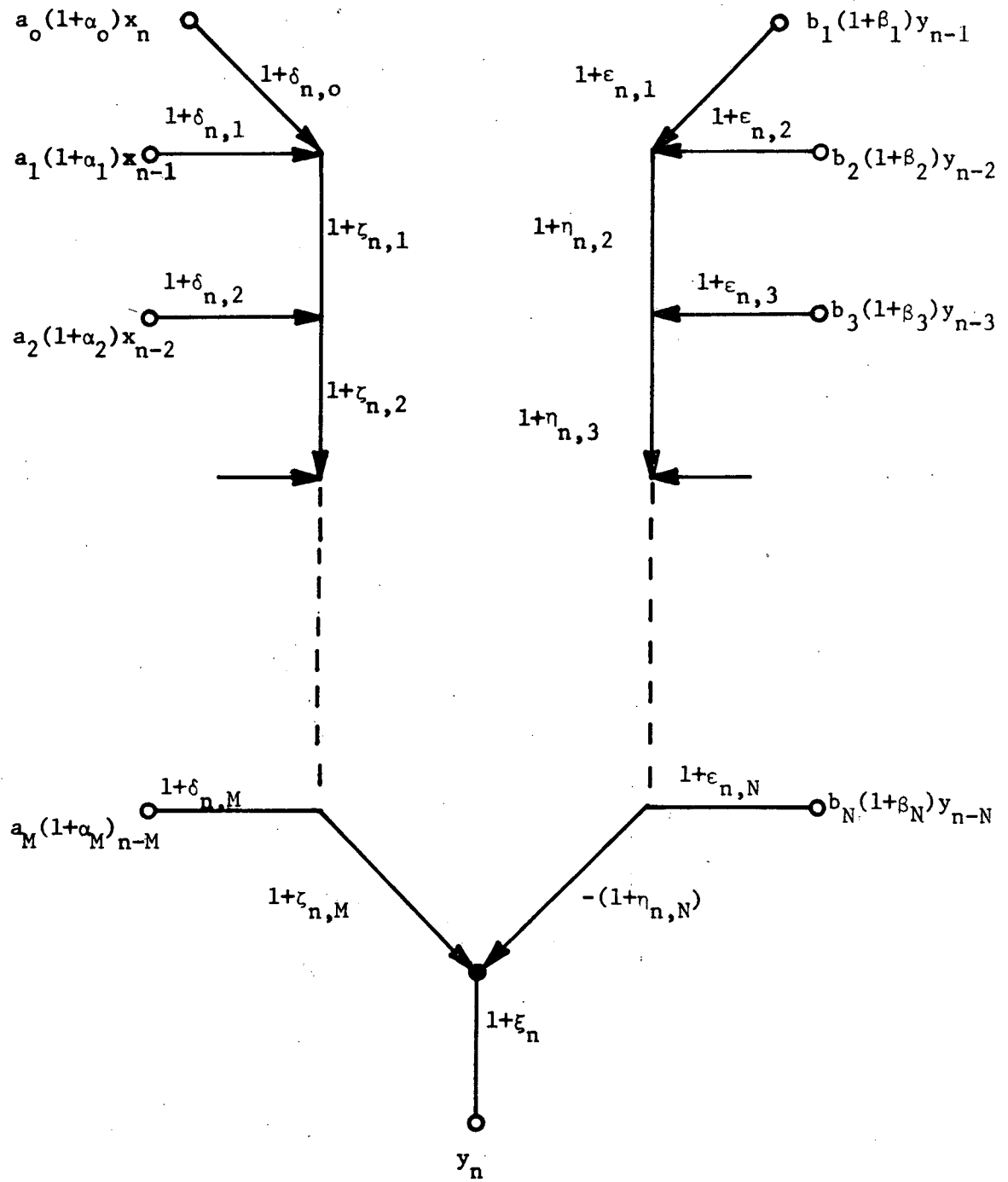


Fig. 25. Flowgraph of Equation (2-81).

The quantities $\delta_{n,k}$; $\zeta_{n,k}$; $\eta_{n,k}$; $\epsilon_{n,k}$; and ξ_n are the errors introduced at each arithmetic step and they are independent random variables uniformly distributed in $[-2^{-t}, 2^{-t}]$.

From Equations (5-1) and (5-12) it can be shown that the error e_n satisfies the following equation:

$$\sum_{k=0}^N b_k e_{n-k} = u'_n + u''_n \quad (5-14)$$

where $b_0 = 1$ and

$$\begin{aligned} u'_n &= \sum_{k=0}^M a_k \alpha_k x_{n-k} - \sum_{k=1}^N b_k \beta_k w_{n-k} \\ u''_n &= \sum_{k=0}^M a_k (\theta_{n,k} - 1) x_{n-k} - \sum_{k=1}^N b_k (\phi_{n,k} - 1) w_{n-k} \end{aligned} \quad (5-15)$$

In the above equations u'_n is due to coefficient rounding; u''_n is due to roundoff accumulations and the input x_n is zero mean and w.s.s. Both components u'_n and u''_n have zero mean and are w.s.s., and they are uncorrelated, this being because $\theta_{n,k}$ and $\phi_{n,k}$ have a mean equal to 1 and are independent of x_n and w_n .

From Equation (5-14), the error sequence e_n is zero mean and w.s.s. with a power spectral density related to those of u'_n and u''_n by

$$\phi_{ee}(z) = [1/(D(z)D(1/z))] [\phi_{u'u'}(z) + \phi_{u''u''}(z)]. \quad (5-16)$$

$\phi_{u'u'}(z)$ is calculated from Equation (5-15) and is given by

$$\phi_{u'u'}(z) = [B(z) - H(z)A(z)] \cdot [B(1/z) - H(1/z)A(1/z)] \phi_{xx}(z) \quad (5-17)$$

where $H(z)$ (Equation (5-6)) is as previously defined and

$$\begin{aligned} A(z) &= \sum_{k=1}^N b_k \beta_k z^{-k} \\ B(z) &= \sum_{k=0}^M a_k \alpha_k z^{-k} \end{aligned} \quad (5-18)$$

Concluding from Equations (5-13), and (5-15), u_n'' is white noise with power spectral density as follows;

$$\begin{aligned} \phi_{u''u''}(z) &= q^{2/2\pi j} \oint (F(z) + G(z)H(z)H(1/z) \\ &\quad - N(1/z)[D(z) - 1]H(z) \\ &\quad - N(z)[D(1/z) - 1]H(1/z)) \phi_{xx}(z)/z \, dz \end{aligned} \quad (5-19)$$

where $N(z) = \sum_{k=0}^M a_k z^{-k}$ is the numerator of the transfer function in Equation (5-6) and

$$F(z) = \sum_{k=0}^M \sum_{i=0}^M a_k a_i F_{k,i} z^{k-i}$$

$$G(z) = \sum_{k=1}^N \sum_{i=1}^N b_k a_i G_{i,i} z^{k-i}$$

$$F_{k,i} = \begin{cases} M + 2 - \max(k,i), & k \neq i \text{ or } k = i = 0 \\ M + 3 - k, & k = i \neq 0 \end{cases}$$

$$G_{k,i} = \begin{cases} N + 2 - \max(k,i), & k \neq i \text{ or } k = i = 1 \\ N + 3 - k, & k = i \neq 1 \end{cases} \quad (5-20)$$

The mean squared value of the error e_n can now be calculated from $\phi_{ee}(z)$ by using

$$E\{e_n^2\} = 1/2\pi j \oint \phi_{ee}(z)/z \, dz. \quad (5-21)$$

VI. PROGRAMMING FORMS

The structure of a digital filter is described by a unique set of constant coefficient linear difference equations. These difference equations constitute the digital filter's programming form. As a general rule, for any programming form the lower the order n of the filter transfer function the smaller the error introduced into the system by coefficient and signal amplitude quantization. Consequently, a n^{th} order filter is usually factored into second-order modules which are paralleled or cascaded to realize the higher orders. The second-order is chosen so that complex poles and zeroes are realizable.

The z -transfer function for any second-order module may be expressed

$$D(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \quad (6-1)$$

The eleven programming forms presented here will be for the second-order module of equation (6-1). For a higher-order digital filter, the following procedure applies: 1) Section $D(z)$ into second-order modules, 2) analyze each module using the computer-aided design procedure to be developed later, and 3) cascade (or parallel) the resulting designs to realize the original $D(z)$.

This section will summarize, for equation (6-1), eleven different programming forms and the attributes of each needed for quantization analysis

by steady-state, statistical, and upper-bound techniques. In particular, the transfer function $T_j(z)$ from the j^{th} quantizer to the filter output for equation (4-9) and (4-18), and the discrete-time difference equations necessary for the impulse response from the j^{th} quantizer to the filter output for equations (4-11), (4-19), and (4-32), will be listed for each programming form. The tabulation of the eleven programming forms is a result of [38, 63-66]. Many others are possible as seen in [67-70,76].

Direct Form

The direct programming form for equation (6-1) is shown in Fig. 26. This form has an A/D or input quantizer, Q_1 , digital-to-analog (D/A) or output quantizer Q_2 and one internal feedback quantizer Q_3 . The transfer functions from each quantizer to the output are

$$T_1(z) = D(z)$$

$$T_2(z) = 1 \tag{6-2}$$

$$T_3(z) = - \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} .$$

The integrands for equation (4-18) are thus

$$\frac{T_1(z) T_1(1/z)}{z} = \frac{(a_0 z^2 + a_1 z + a_2)(a_0 + a_1 z + a_2 z^2)/b_2}{z(z^2 + b_1 z + b_2)(z^2 + b_1 z/b_2 + 1/b_2)}$$

$$\frac{T_2(z) T_2(1/z)}{z} = \frac{1}{z} \tag{6-3}$$

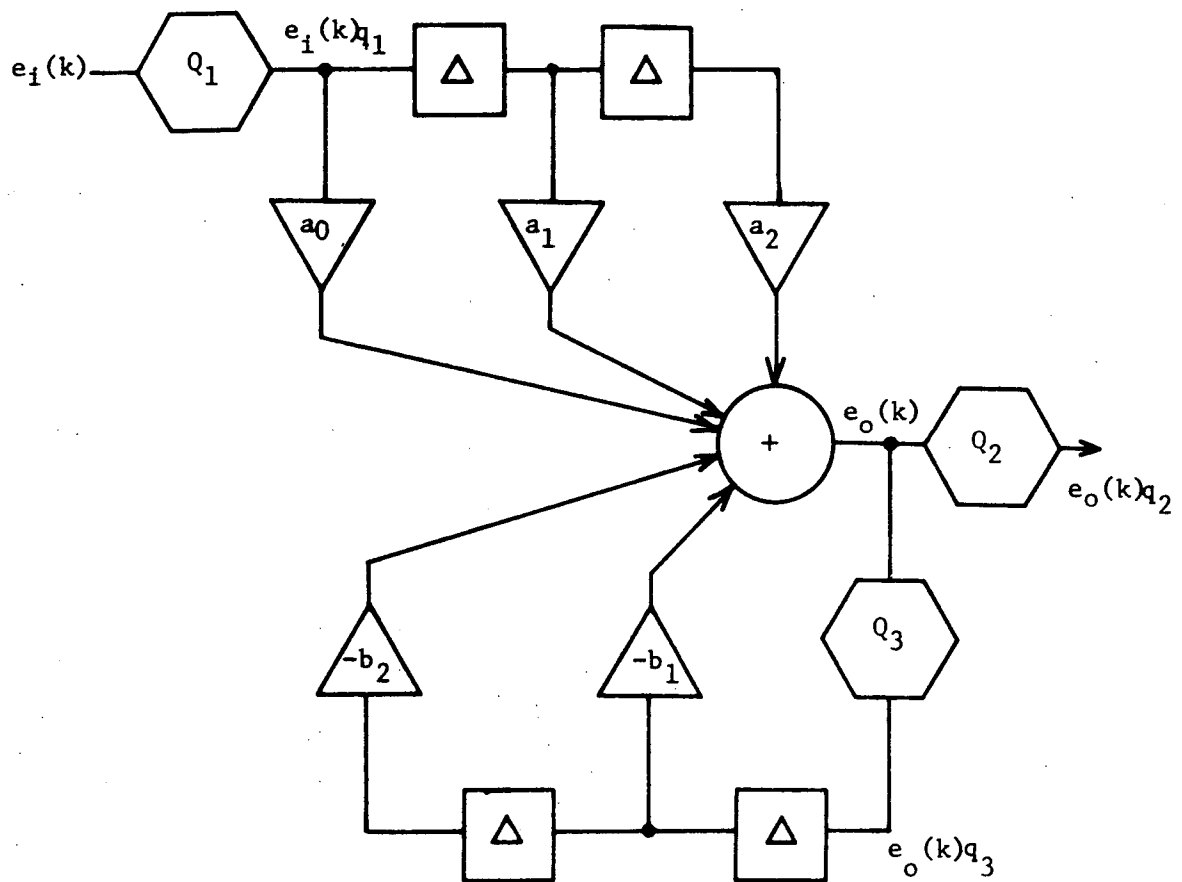


Fig 26. The Direct Form.

$$\frac{T_3(z) T_3(1/z)}{z} = \frac{(b_1 z + b_2)(b_1 z + b_2 z^2)/b_2}{z(z^2 + b_1 z + b_2)(z^2 + b_1 z/b_2 + 1/b_2)}.$$

For programming and impulse testing the difference equations for the direct form are

$$e_1(k)_{q_1} = e_i(k) + n_1(k)$$

$$e_o(k) = a_0 e_1(k)_{q_1} + a_1 e_1(k-1)_{q_1} + a_2 e_1(k-2)_{q_1}$$

$$-b_1 e_o(k-1)_{q_3} - b_2 e_o(k-1)_{q_3} \quad (6-4)$$

$$e_o(k)_{q_2} = e_o(k) + n_2(k)$$

$$e_o(k)_{q_3} = e_o(k) + n_3(k).$$

The filter output variable is $e_o(k)_{q_2}$. This completes the description of the direct programming form.

For all eleven programming forms the standard notation of Q_1 for the filter input quantizer and Q_2 for the filter output quantizer has been adopted for convenience. The transfer functions $T_1(z)$ and $T_2(z)$ are then always to be for the input and output quantizers respectively. These transfer functions will be identical for all the programming forms as given in equation (6-2).

Modified Direct Form

The modified direct programming form for equation (6-1) is shown in Fig. 27. This form differs from the direct form only in the feedback

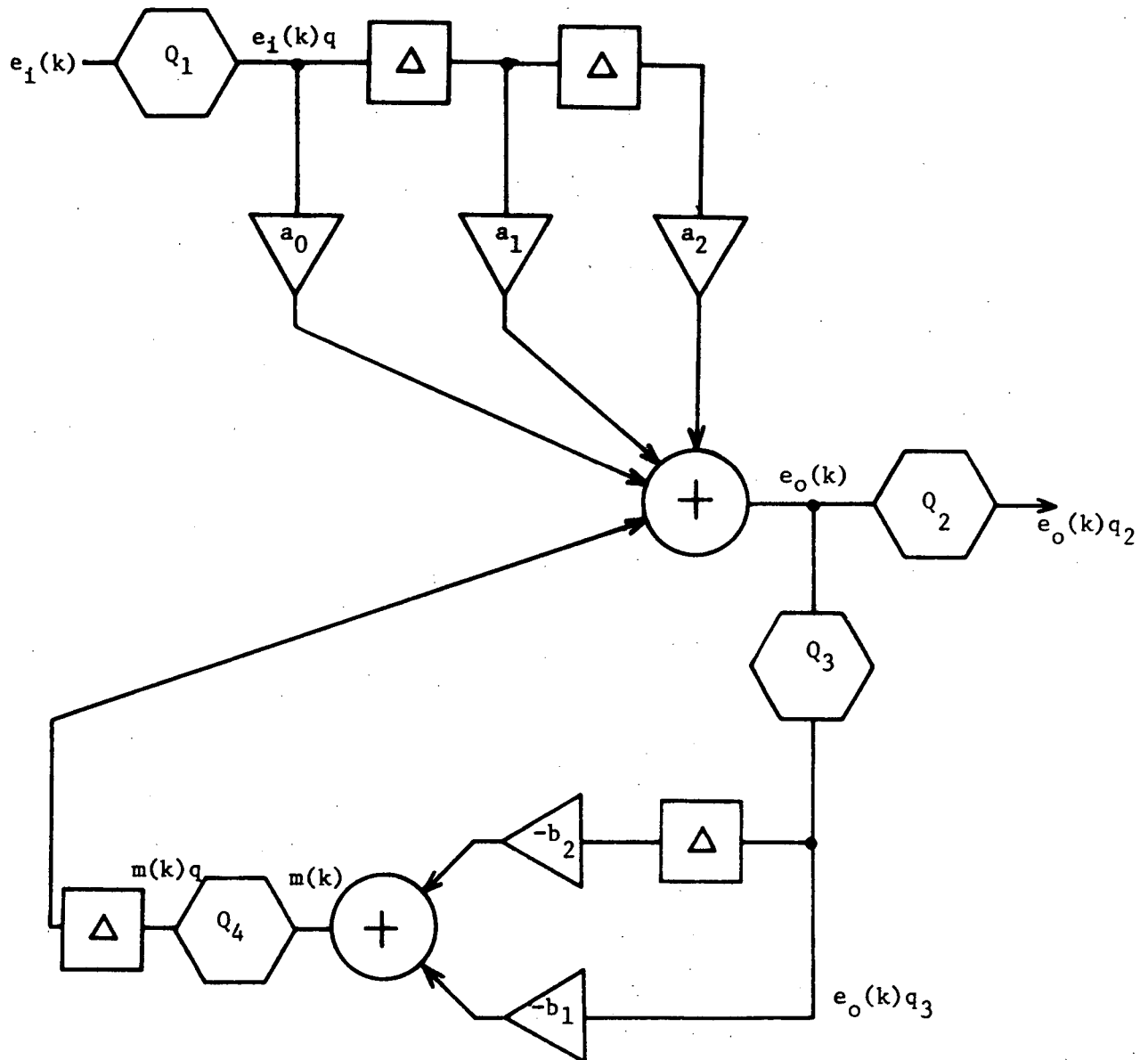


Fig. 27. The Modified Direct Form.

loop. This form has two internal quantizers; Q_3 is identical to the direct form hence $T_3(z)$ is given by equation (6-2); Q_4 has been added and its transfer function to the filter output is displayed below:

$$T_4(z) = \frac{z^{-1}}{1 + b_1 z^{-1} + b_2 z^{-1}} \quad (6-5)$$

The integrand for equation (4-18) for Q_4 becomes

$$\frac{T_4(z) T_4(1/z)}{z} = \frac{z^2/b_2}{z(z^2 + b_1 z + b_2)(z^2 + b_1 z/b_2 + 1/b_2)} \quad (6-6)$$

For programming and impulse testing the difference equations for the modified direct programming form are

$$e_i(k)_q = e_i(k) + n_1(k)$$

$$e_o(k) = a_0 e_i(k)_q + a_1 e_i(k-1)_q + a_2 e_i(k-2)_q \\ + m(k-1)_q$$

$$e_o(k)_{q_2} = e_o(k) + n_2(k) \quad (6-7)$$

$$e_o(k)_{q_3} = e_o(k) + n_3(k)$$

$$m(k) = -b_1 e_o(k)_{q_3} - b_2 e_o(k-1)_{q_3}$$

$$m(k)_q = m(k) + n_4(k).$$

Standard Form

The standard programming form for equation (6-1) is presented in Fig. 28. This form has two internal quantizers, Q_3 and Q_4 . Their transfer functions to the filter output are

$$\begin{aligned} T_3(z) &= \frac{1}{z^2 + b_1 z + b_2} \\ T_4(z) &= \frac{z + b_1}{z^2 + b_1 z + b_2} \end{aligned} \quad (6-8)$$

The integrands for equation (4-18) are

$$\begin{aligned} \frac{T_3(z) T_3(1/z)}{z} &= \frac{z^2/b_2}{z(z^2 + b_1 z + b_2)(z^2 + b_1 z/b_2 + 1/b_2)} \\ \frac{T_4(z) T_4(1/z)}{z} &= \frac{(z + b_1)(z + b_1 z^2)/b_2}{z(z^2 + b_1 z + b_2)(z^2 + b_1 z/b_2 + 1/b_2)} \end{aligned} \quad (6-9)$$

The difference equations for this form are

$$\begin{aligned} e_i(k)_q &= e_i(k) + n_1(k) \\ e_o(k) &= a_0 e_i(k)_q + m_2(k-1)_q \\ e_o(k)_q &= e_o(k) + n_2(k) \\ m_1(k) &= \alpha_2 e_i(k)_q - b_1 m_1(k-1)_q - b_2 m_2(k-1)_q \end{aligned} \quad (6-10)$$

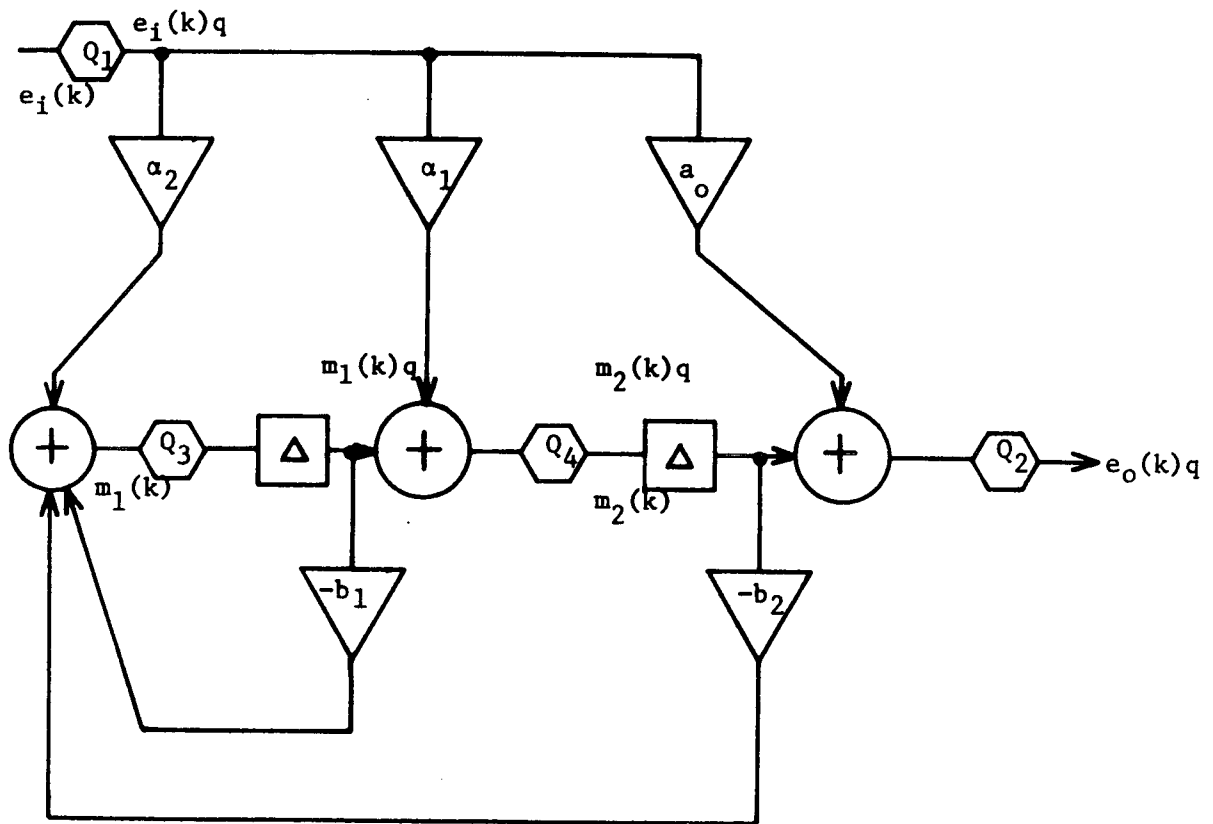


Fig. 28. The Standard Form.

$$m_1(k)_q = m_1(k) + n_3(k)$$

$$m_2(k) = \alpha_1 e_i(k)_q + m_1(k-1)_q$$

$$m_2(k)_q = m_2(k) + n_4(k)$$

where

$$\alpha_1 = a_1 - a_0 b_1$$

$$\alpha_2 = a_2 - a_0 b_2 - b_1 \alpha_1.$$

Modified Standard Form

Again the modified standard form is for $D(z)$ as expressed in equation (6-7) and is demonstrated in Fig. 29. This programming form differs from the standard form in its feedback loops. The same internal quantizers are present as before with a fifth quantizer added. The transfer functions for the three quantizers are

$$T_3(z) = T_3(z) \text{ in (6-2)}$$

$$T_4(z) = T_3(z) \text{ in (6-8)}$$

$$T_5(z) = T_4(z) \text{ in (6-5)}$$

Hence, the integrands for equation (4-18) have been previously shown.

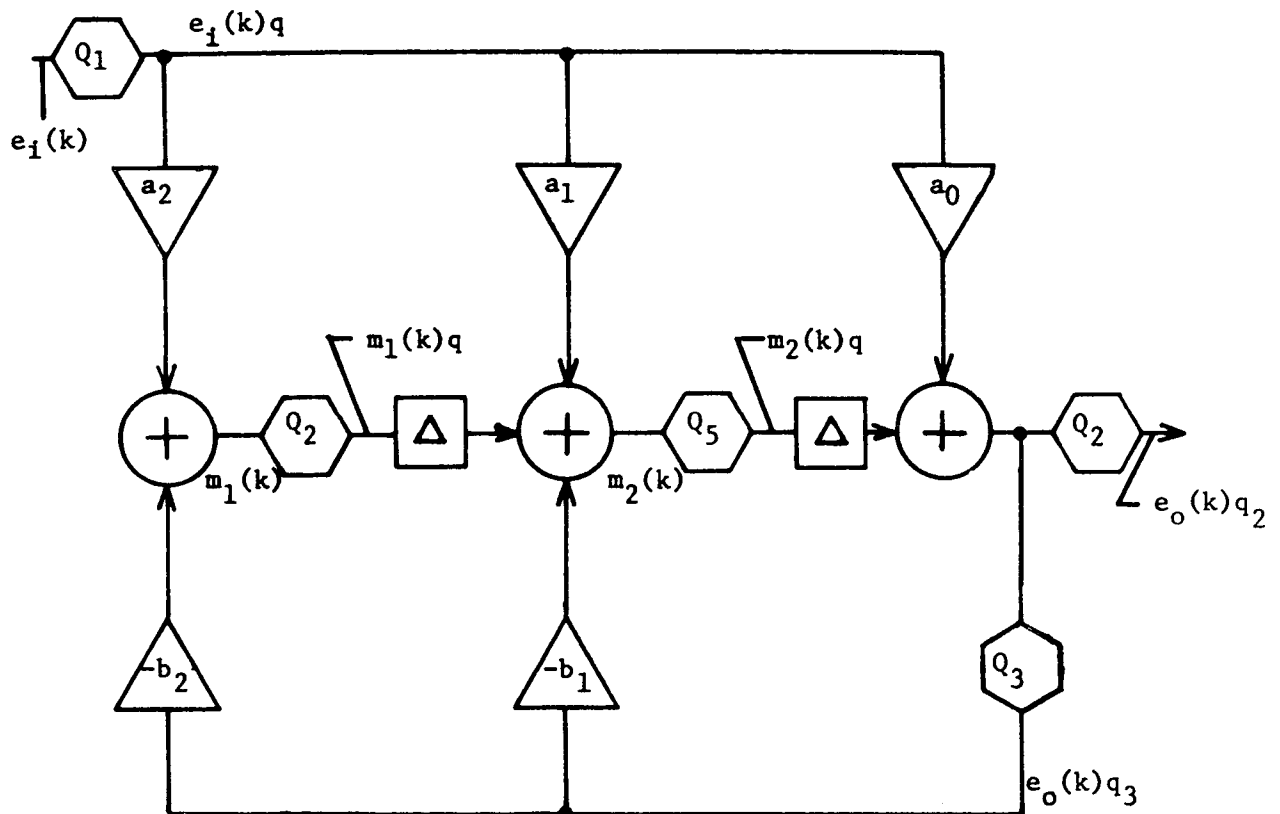


Fig. 29. The Modified Standard Form.

For programming, etc., the difference equations for the modified standard form are

$$e_i(k)_q = e_i(k) + n_1(k)$$

$$e_o(k) = a_0 e_i(k)_q + m_2(k-1)_q$$

$$e_o(k)_{q_2} = e_o(k) + n_2(k)$$

$$e_o(k)_{q_3} = e_o(k) + n_3(k)$$

(6-11)

$$m_1(k) = a_2 e_i(k)_q - b_2 e_o(k)_{q_3}$$

$$m_1(k)_q = m_1(k) + n_4(k)$$

$$m_2(k) = a_1 e_i(k)_q + m_1(k-1)_q - b_1 e_o(k)_{q_3}$$

$$m_2(k)_q = m_2(k) + n_5(k).$$

Canonical Form

The block diagram for the canonical programming form limited to the second-order module of equation (6-1) is shown in Fig. 30. This form has only one quantizer Q_3 whose transfer function to the filter output is given by

$$T_3(z) = D(z).$$

Therefore, Q_3 has the same effect as the input quantizer on the filter output. The difference equations including quantization are shown below:

$$\begin{aligned}
 e_i(k)_q &= e_i(k) + n_1(k) \\
 m(k) &= e_i(k)_q - b_1 m(k-1)_q - b_2 m(k-2)_q \\
 m(k)_q &= m(k) + n_3(k) \\
 e_o(k) &= a_0 m(k)_q + a_1 m(k-1)_q + a_2 m(k-2)_q \\
 e_o(k)_q &= e_o(k) + n_2(k).
 \end{aligned} \tag{6-12}$$

Modified Canonical Form

The modified canonical programming form for the second-order $D(z)$ of equation (6-1) is depicted in the block diagram of Fig. 31. This programming form differs from the canonical form by its forward transfer paths. By moving the multiplier coefficient from $m(k)_q$ to $e_i(k)_q$ the transfer function for Q_3 is changed:

$$T_3(z) = \frac{\alpha_1 z + \alpha_2}{z^2 + b_1 z + b_2}, \tag{6-13}$$

where

$$\alpha_1 = a_1 - a_0 b_1$$

$$\alpha_2 = a_2 - a_0 b_2.$$

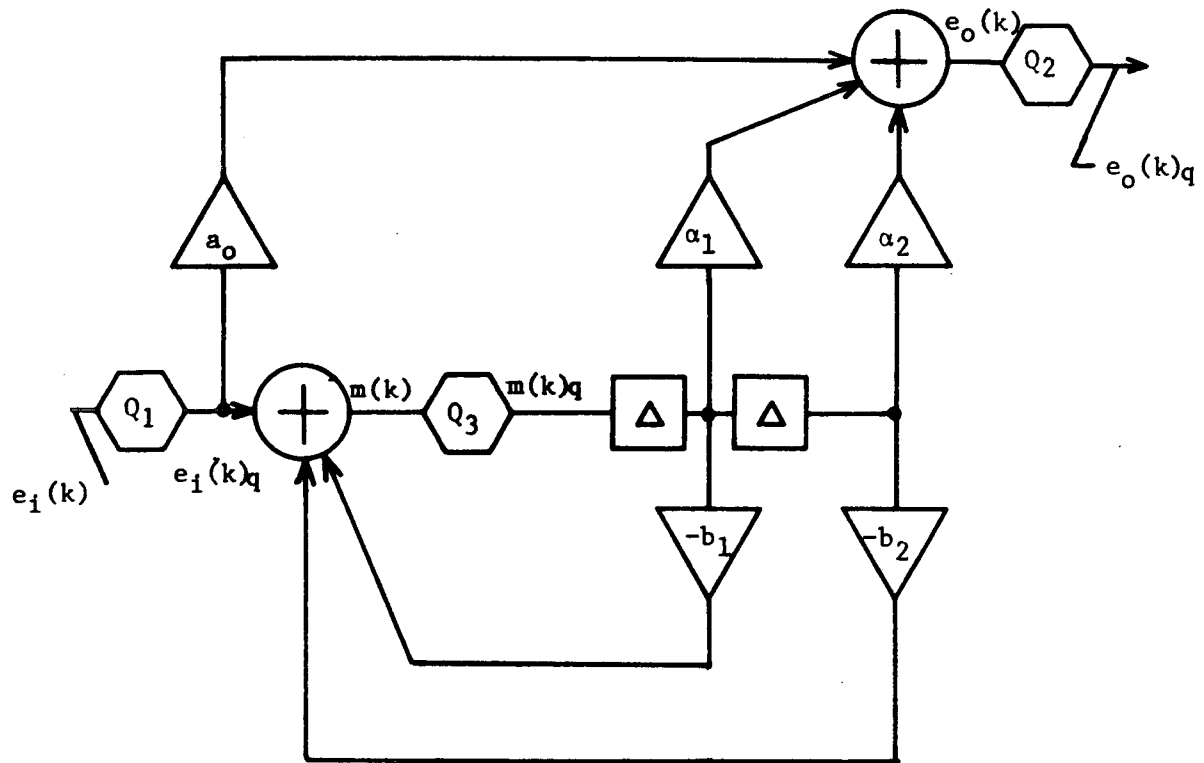


Fig. 31. The Modified Canonical Form.

The integrand for equation (4-18) for this transfer function is

$$\frac{T_3(z) T_3(1/z)}{z} = \frac{(\alpha_1 z + \alpha_2)(\alpha_1 z + \alpha_2 z^2)}{z(z^2 + b_1 z + b_2)(z^2 + b_1 z/b_2 + 1/b_2)} \quad (6-14)$$

The difference equations for the modified canonical programming form are shown below:

$$\begin{aligned} e_i(k)_q &= e_i(k) + n_1(k) \\ e_o(k) &= a_0 e_i(k)_q + \alpha_1 m(k-1)_q + \alpha_2 m(k-2)_q \\ e_o(k)_q &= e_o(k) + n_2(k) \\ m(k) &= e_i(k)_q - b_1 m(k-1)_q - b_2 m(k-2)_q \\ m(k)_q &= m(k) + n_3(k). \end{aligned} \quad (6-15)$$

The six programming forms discussed to this point have all required the programming coefficients a_i and b_i of equation (6-1), or were easily calculated from them. The last five forms which are to be presented now will require more effort to find the correct form for $D(z)$ and the proper programming parameters for the difference equations.

Parallel Form

The general block diagram for the parallel programming form for a second-order $D(z)$ is shown in Fig. 32. The form may be used if and only if the second-order module has real poles p_1 and p_2 . Hence, $D(z)$ must have the form

$$D(z) = a_0 + \frac{R_1}{z-p_1} + \frac{R_2}{z-p_2} . \quad (6-16)$$

The constants R_1 and R_2 are real numbers representing the residues of poles p_1 and p_2 , and p_1 should be different from p_2 .

The coefficients g_i shown in Fig. 32 must satisfy the following relationships:

$$\begin{aligned} g_1 g_2 &= R_1 \\ g_3 g_4 &= R_2 \end{aligned} \quad (6-17)$$

In order to minimize the magnitude of the parameters g_i the following choices were arbitrarily made:

$$\begin{aligned} g_1 &= \sqrt{|R_1|} \\ g_2 &= R_1/g_1 \\ g_3 &= \sqrt{|R_2|} \\ g_4 &= R_2/g_3 . \end{aligned} \quad (6-18)$$

The transfer functions from the internal quantizers to the filter output were obtained:

$$T_3(z) = \frac{g_2}{z-p_1} \quad (6-19)$$

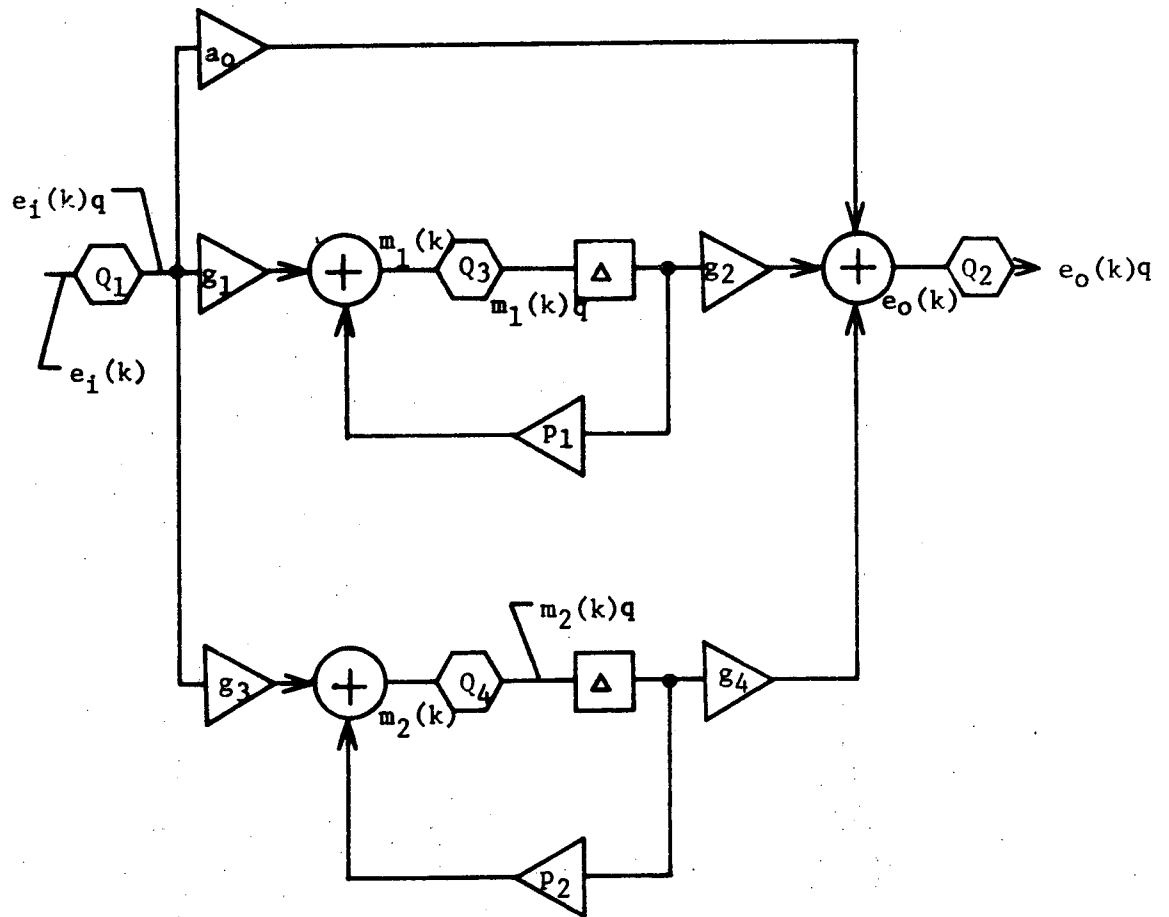


Fig. 32. The Parallel Form.

$$T_4(z) = \frac{g_4}{z-p_2}.$$

The integrands for equation (4-18) corresponding to (6-19) are

$$\frac{T_3(z)T_3(1/z)}{z} = \frac{-g_2^2 z/p_1}{z(z-p_1)(z-1/p_1)} \quad (6-20)$$

$$\frac{T_4(z)T_4(1/z)}{z} = \frac{-g_4^2 z/p_2}{z(z-p_2)(z-1/p_2)}.$$

The difference equations for the parallel form are

$$\begin{aligned} e_i(k)_q &= e_i(k) + n_1(k) \\ e_o(k) &= a_0 e_i(k)_q + g_2 m_1(k-1)_q + g_4 m_2(k-1)_q \\ e_o(k)_q &= e_o(k) + n_2(k) \\ m_1(k) &= g_1 e_i(k)_q + p_1 m_1(k-1)_q \\ m_1(k)_q &= m_1(k) + n_3(k) \\ m_2(k) &= g_3 e_i(k)_q + p_2 m_2(k-1)_q \\ m_2(k)_q &= m_2(k) + n_4(k). \end{aligned} \quad (6-21)$$

Please note that the parallel form can realize only real poles, but it is capable of realizing either real or complex zeroes.

Cascade Form

The cascade programming form for a second-order digital filter module essentially factors the module into first order stages and realizes each stage individually. If each first order stage is implemented in the manner of Fig. 30; the resulting cascade form is

shown in Fig. 33. A requirement for this form is that

$$D(z) = a_0 \frac{(z-q_1)(z-q_2)}{(z-p_1)(z-p_2)}, \quad (6-22)$$

where q_i and p_i are real zeroes and poles. Also, the following relationships must be satisfied:

$$\begin{aligned} a_0 &= g_1 g_2 g_4 \\ g_3 &= -g_1 g_2 \\ g_5 &= -g_2 g_4 \end{aligned} \quad (6-23)$$

The cascade form has two internal quantizers which are described by the transfer functions

$$T_3(z) = D(z)/g_1 \quad (6-24)$$

$$T_4(z) = g_4 \frac{z-q_2}{z-p_2}$$

The integrands for equation (4-18) are

$$\frac{T_3(z) T_3(1/z)}{z} = \frac{1}{g_1^2} \frac{D(z) D(1/z)}{z} \quad (6-25)$$

$$\frac{T_4(z) T_4(1/z)}{z} = \frac{-g_4^2 (z-q_2)(1-q_2 z)/p_2}{z(z-p_2)(z-1/p_2)}.$$

The difference equations for this cascade form are displayed below:

$$e_i(k)_q = e_i(k) + n_1(k) \quad (6-26)$$

$$m_1(k) = g_1 e_i(k)_q + p_1 m_1(k-1)_q$$

$$m_1(k)_q = m_1(k) + n_3(k)$$

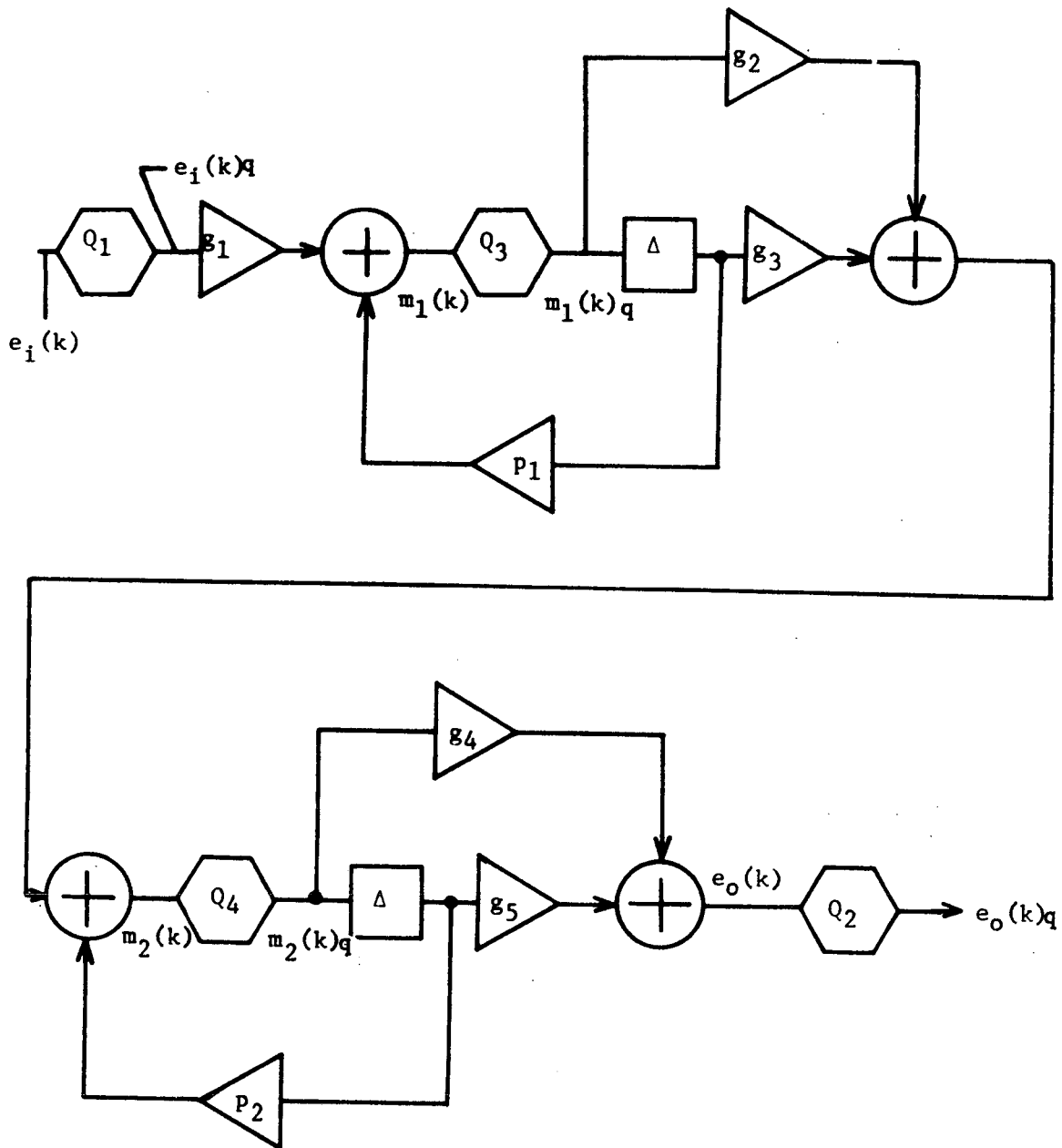


Fig. 33. The Cascade Form.

$$m_2(k) = g_2 m_1(k)_q + g_3 m_1(k-1)_q + p_2 m_2(k-1)_q$$

$$m_2(k)_q = m_2(k) + n_4(k)$$

$$e_o(k) = g_4 m_2(k)_q + g_5 m_2(k-1)_q$$

$$e_o(k)_q = e_o(k) + n_2(k) \quad .$$

The parameters g_i , $i=1, 5$ in equation (6-26) must be found using (6-23). Since there are three equations with five unknowns, an arbitrary choice for g_1 and g_2 is made as follows:

$$\begin{aligned} g_1 &= 1.0 \\ g_2 &= \sqrt{|a_o|}. \end{aligned} \tag{6-27}$$

If a_o is zero, this form cannot be realized.

This completes the cascade form. In summary, this programming form is applicable to a second-order digital filter module when it is possible to cascade first-order stages programmed in the canonical form.

Modified Cascade Form

Of the many possible ways of implementing first-order stages, one other technique was selected which employs the modified canonical form for each first-order section (see Fig. 34). This programming form is labeled modified cascade; it requires $D(z)$ to be factorable into real poles and zeroes as in equation (6-22).

The transfer functions from the three internal quantizers to the filter output are given below:

$$T_3(z) = \frac{g_3 g_4 (z - q_2)}{(z - p_1)(z - p_2)} \tag{6-28}$$

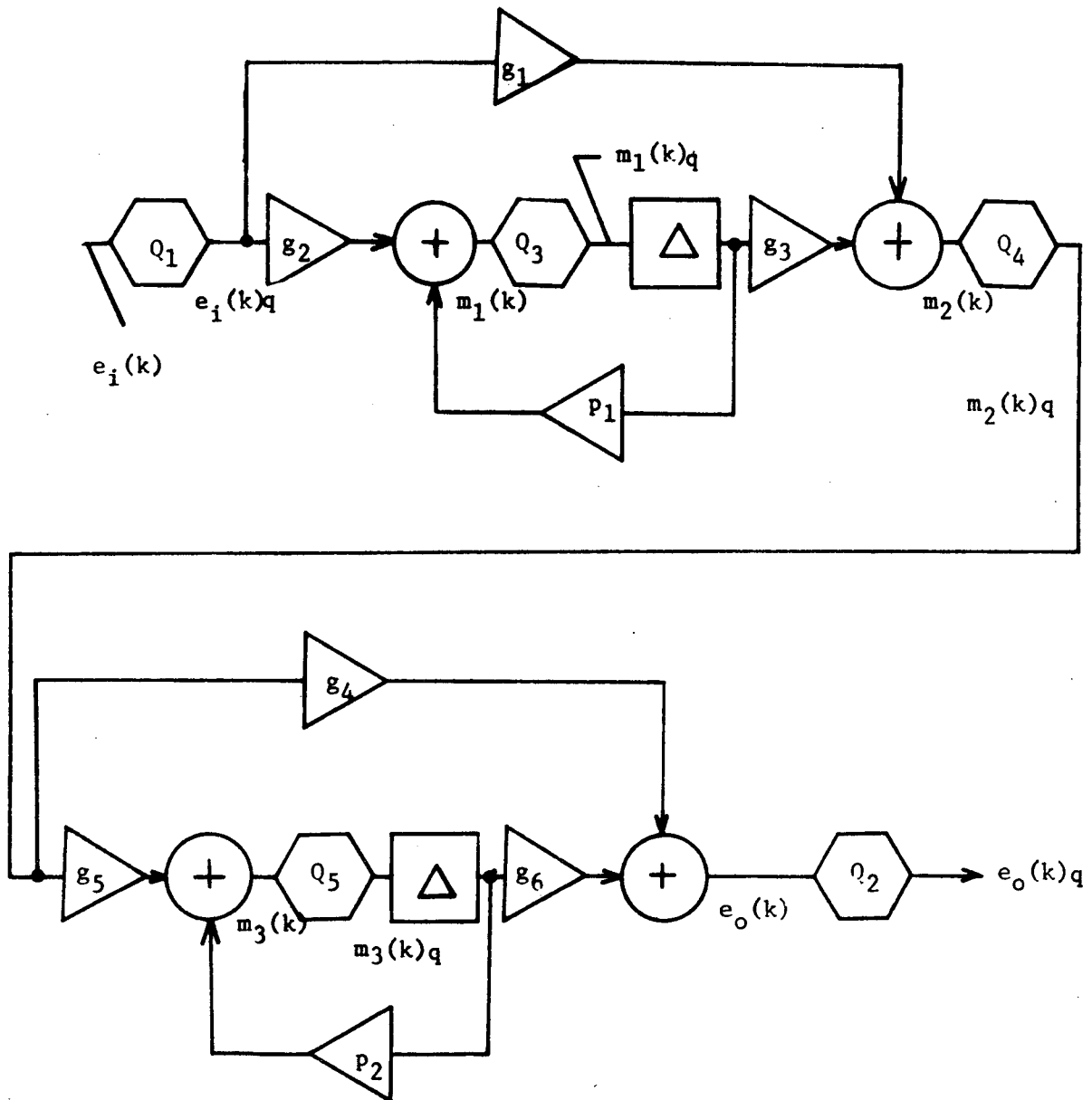


Fig. 34. The Modified Cascade Form.

$$T_4(z) = g_4 \frac{z^{-q_2}}{z^{-p_2}}$$

$$T_5(z) = \frac{g_6}{z^{-p_2}},$$

where the parameters g_i are restricted by

$$\begin{aligned} g_1 g_4 &= a_0 \\ g_2 g_3 &= (p_1 - q_1) g_1 \\ g_5 g_6 &= (p_2 - q_2) g_4 \end{aligned} \quad (6-29)$$

Since there are three equations and six unknowns, arbitrary choices are again made for g_1 , g_2 , and g_4 as follows:

$$\begin{aligned} g_1 &= \sqrt{|a_0|} \\ g_2 &= (p_1 + 1)/2 \\ g_3 &= (p_1 - q_1) g_1 / g_2 \\ g_4 &= a_0 / g_1 \\ g_5 &= (p_2 + 1)/2 \\ g_6 &= (p_2 - q_2) g_4 / g_5 \end{aligned} \quad (6-30)$$

Using these parameters, the following difference equations may be used to implement this programming form:

$$\begin{aligned} e_i(k)_q &= e_i(k) + n_1(k) \\ m_2(k) &= g_1 e_i(k)_q + g_3 m_1(k-1)_q \\ m_2(k)_q &= m_2(k) + n_4(k) \\ e_o(k) &= g_4 m_2(k) + g_6 m_3(k-1)_q \\ e_o(k)_q &= e_o(k) + n_2(k) \\ m_1(k) &= g_2 e_i(k)_q + p_1 m_1(k-1)_q \\ m_1(k)_q &= m_1(k) + n_3(k) \end{aligned} \quad (6-31)$$

$$m_3(k) = g_5 m_2(k)_q + p_2 m_3(k-1)_q$$

$$m_3(k)_q = m_3(k) + n_5(k) .$$

X1 Structure

The last two programming forms to be presented are designed for a second-order $D(z)$ with complex poles. The appropriate expression for the transfer function is

$$D(z) = a_0 + \frac{A}{z+p} + \frac{A^*}{z+p^*}, \quad (6-32)$$

where a_0 has been previously defined, p and p^* are complex conjugate poles, and A and A^* are complex conjugate residues.

The first implementation of (6-32) is depicted in the block diagram of Fig. 35. The parameters indicated in the figure are defined below:

$$g_1 = -\text{Re}(p)$$

$$g_2 = \text{Im}(p)$$

$$g_3 = 2 \text{ Im}(A)$$

$$g_4 = 2 \text{ Re}(A) .$$
(6-33)

The two internal quantizers, Q_3 and Q_4 , are described by the transfer functions

$$T_3(z) = \frac{g_2}{z^2 + b_1 z + b_2} \quad (6-34)$$

$$T_4(z) = \frac{z^{-g_1}}{z^2 + b_1 z + b_2} .$$

The corresponding integrands for equation (4-18) are

$$\frac{T_3(z) T_3(1/z)}{z} = \frac{g_2^2 z^2 / b_2}{z(z^2 + b_1 z + b_2) (z^2 + b_1 z / b_2 + 1/b_2)} \quad (6-35)$$

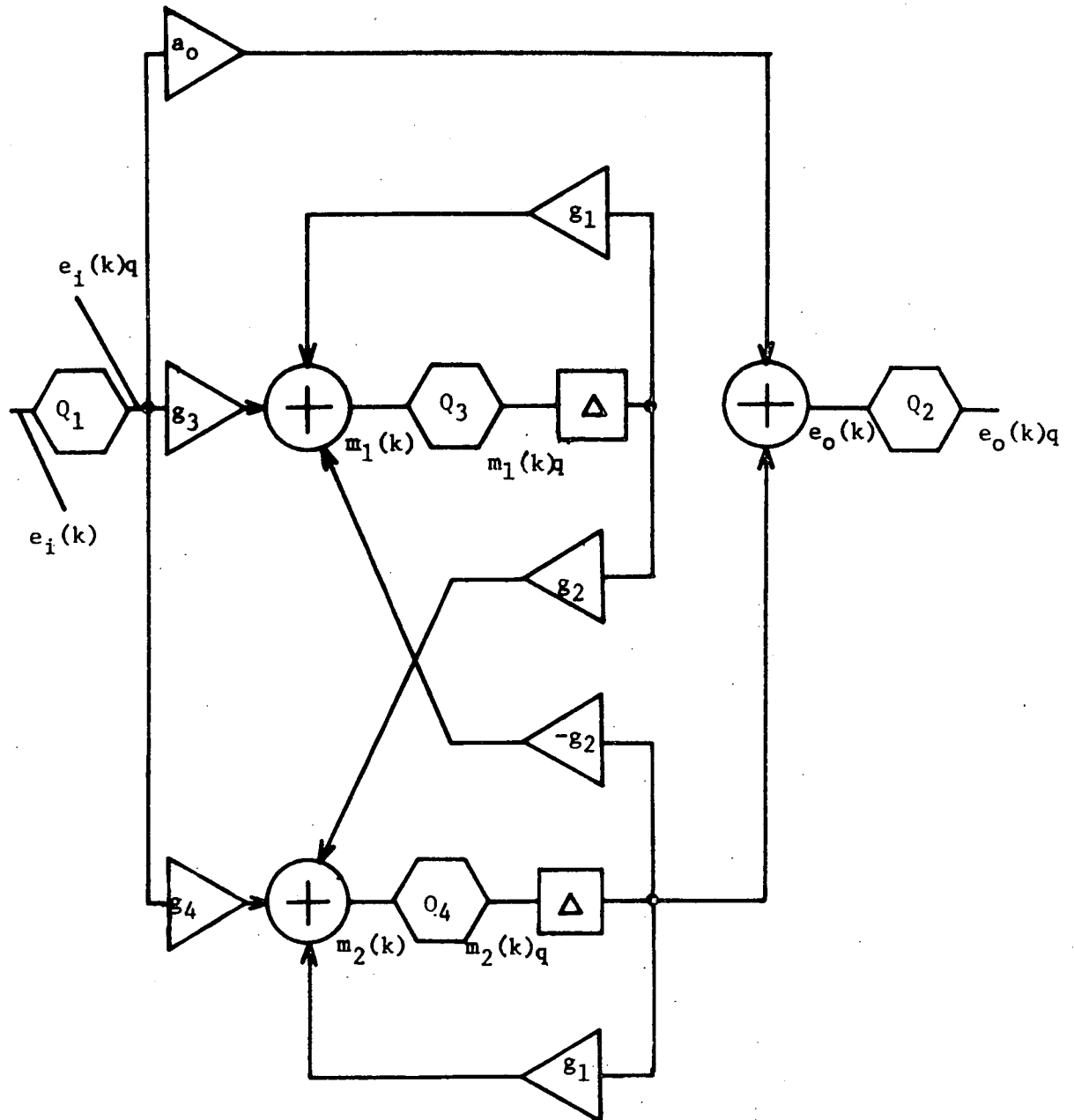


Fig. 35. The XI Structure.

$$\frac{T_4(z) T_4(1/z)}{z} = \frac{(z-g_1) (z-g_1 z^2)/b_2}{z(z^2+b_1 z+b_2) (z^2+b_1 z/b_2+1/b_2)} .$$

The difference equations for the XI structure are enumerated below:

$$\begin{aligned} e_i(k)_q &= e_i(k) + n_1(k) \\ e_o(k) &= a_o e_i(k)_q + m_2(k-1)_q \\ e_o(k)_q &= e_o(k) + n_2(k) \\ m_1(k) &= g_3 e_i(k)_q + g_1 m_1(k-1)_q - g_2 m_2(k-1)_q \\ m_1(k)_q &= m_1(k) + n_3(k) \\ m_2(k) &= g_4 e_i(k)_q + g_1 m_2(k-1)_q + g_2 m_1(k-1)_q \\ m_2(k)_q &= m_2(k) + n_4(k) . \end{aligned} \tag{6-36}$$

X2 Structure

The last programming form presented in this paper is the X2 structure of Fig. 36. The transfer function $D(z)$ must be expressed in the format of equation (6-32) in order to use this form.

This programming form has two internal quantizers whose transfer functions to the filter output are

$$T_3(z) = \frac{g_3 z + (g_2 g_4 - g_1 g_3)}{z^2 + b_1 z + b_2} \tag{6-37}$$

$$T_4(z) = \frac{g_4 z - (g_2 g_3 + g_1 g_4)}{z^2 + b_1 z + b_2} ,$$

where

$$\begin{aligned} g_1 &= -\text{Re} (p) \\ g_2 &= \text{Im} (p) \\ g_3 &= -\text{Im} (A) \\ g_4 &= \text{Re} (A) . \end{aligned} \tag{6-38}$$

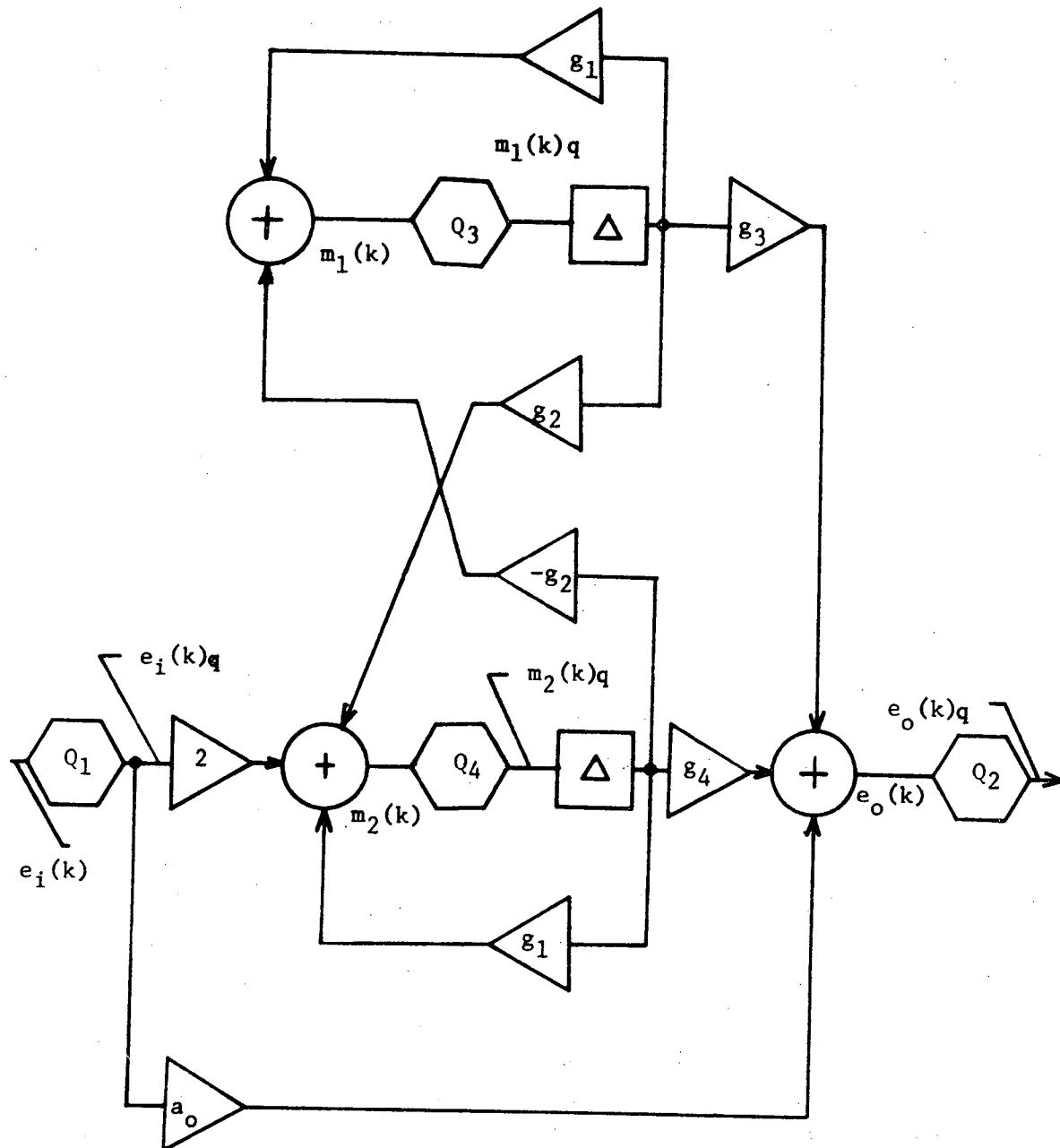


Fig. 36. The X2 Structure

The integrands for equation (4-18) are

$$\frac{T_3(z)T_3(1/z)}{z} = \frac{(g_3z+\delta_1)(g_3z+\delta_1z^2)/b_2}{z(z^2+b_1z+b_2)(z^2+b_1z/b_2+1/b_2)} \quad (6-39)$$

$$\frac{T_4(z)T_4(1/z)}{z} = \frac{(g_4z+\delta_2)(g_4z+\delta_2z^2)/b_2}{z(z^2+b_1z+b_2)(z^2+b_1z/b_2+1/b_2)},$$

where

$$\delta_1 = g_2g_4 - g_1g_3$$

$$\delta_2 = -g_2g_3 - g_1g_4.$$

The difference equations for the X2 structure are listed below:

$$\begin{aligned} e_i(k)_q &= e_i(k) + n_1(k) \\ e_o(k) &= a_0e_i(k)_q + g_3m_1(k-1)_q + g_4m_2(k-1)_q \\ e_o(k)_q &= e_o(k) + n_2(k) \\ m_1(k) &= g_1m_1(k-1)_q - g_2m_2(k-1)_q \\ m_1(k)_q &= m_1(k) + n_3(k) \\ m_2(k) &= 2e_i(k)_q + g_1m_2(k-1)_q + g_2m_1(k-1)_q \\ m_2(k)_q &= m_2(k) + n_4(k). \end{aligned} \quad (6-40)$$

This completes the X2 structure.

Summary of Programming Forms

This section has summarized the essential characteristics of eleven programming forms for a second-order digital filter module. All of the equations necessary to perform steady-state, statistical, and error bound analyses have been determined. A pattern may be observed in the formats of the relations for equation (4-18), the

residue evaluation equation for statistical analysis. All of the integrands fall into the following formats:

$$F_1(z) = \frac{\gamma_3(\gamma_0 z^2 + \gamma_1 z + \gamma_2) (\gamma_0 + \gamma_1 z + \gamma_2 z^2)}{z(z^2 + b_1 z + b_2) (z^2 + b_1 z/b_2 + 1/b_2)} \quad (6-41)$$

or

$$F_2(z) = \frac{\gamma_2(z - \gamma_1) (1 - \gamma_1 z)}{z(z - \gamma_0) (z - 1/\gamma_0)} .$$

Table 2 displays the respective equations for each programming form using equation (6-41).

Many other characteristics of each programming form should also be investigated; for example, the coefficient sensitivity and the deadband effects are also important for good digital filter operation.

Table 2. Integrands for (4-18).

Programming Form	Quantizer	Format	Parameters			
			γ_0	γ_1	γ_2	γ_3
All	Q_1	F_1	a_0	a_1	a_2	$1/b_2$
	Q_2	-				
Direct	Q_3	F_1	0	b_1	b_2	$1/b_2$
Modified Direct	Q_3	F_1	0	b_1	b_2	$1/b_2$
	Q_4	F_1	0	0	1	$1/b_2$
Standard	Q_3	F_1	0	0	1	$1/b_2$
	Q_4	F_1	0	1	b_1	$1/b_2$
Modified Standard	Q_3	F_1	0	b_1	b_2	$1/b_2$
	Q_4	F_1	0	0	1	$1/b_2$
	Q_5	F_1	0	0	1	$1/b_2$
Canonical	Q_3	F_1	a_0	a_1	a_2	$1/b_2$
Modified Canonical	Q_3	F_1	0	α_1	α_2	$1/b_2$
Parallel	Q_3	F_2	p_1	0	$-g_2^2/p_1$	
	Q_4	F_2	p_2	0	$-g_4^2/p_1$	
Cascade	Q_3	F_1	a_0	a_1	a_2	$1/g_1^2 b_2$
	Q_4	F_2	p_2	q_2	$-g_4^2/p_2$	
Modified Cascade	Q_3	F_1	0	1	$-q_2$	$g_3^2 g_4^2 / p_1 p_2$
	Q_4	F_2	p_2	q_2	$-g_4^2/p_2$	
	Q_5	F_2	p_2	0	$-g_6^2/p_2$	

Table 2. Integrands for (4-18). (Cont'd)

Programming Form	Quantizer	Format	Parameters			
			γ_0	γ_1	γ_2	γ_3
X1 Structure	Q_3	F_1	0	0	1	g_2^2/b_2
	Q_4	F_1	0	1	$-g_1$	$1/b_2$
X2 Structure	Q_3	F_1	0	g_3	δ_1	$1/b_2$
	Q_4	F_1	0	g_4	δ_2	$1/b_2$

VII. COMPUTER AIDED DESIGN

In the design of complex system, the digital computer serves as an essential tool in synthesis and design verification. Computer aided design (CAD) programs are effectively employed in the synthesis of digital filters in three ways: transfer function synthesis, coefficient quantization, and programming form selection.

Transfer Function Synthesis

The digital computer has been used extensively in the design of digital filter transfer functions [30,71-74]. Nonrecursive designs using linear programming has been implemented by Rabiner [73] while Parks and McClellan [72] using polynomial interpolation techniques. Rabiner et al [30] also used a steepest descent technique to obtain FIR filters with minimax error in selected bands.

Recursive digital filters have been synthesized using sampled data transformations by Golden [71]. Robinson and Robinson [74] have demonstrated a CAD program for taking z-transforms. Steiglitz [75] has used nonlinear optimization techniques to obtain recursive digital filter approximations to arbitrary frequency responses.

Coefficient Quantization

Avenhaus [48] has investigated the effects of coefficient optimization for reducing the coefficient wordlength. A given filter is designed and its coefficients are founded. Then an optimizing search is undertaken to find other sets of coefficients which meet the design criteria with a shorter wordlength.

Much work is left to be done in the proper quantization of digital filter coefficients and CAD will surely play a major role in future developments in this area.

Programming Form Selection

A CAD program, listed in [49], has been developed which analyzes the signal amplitude quantization errors in the eleven programming forms presented in Chapter 6. The program, written in FORTRAN IV, is an aid to implementing digital filters for any application, the only restriction is that the filters be expressable as second order stages as shown in equation (6-1).

General

The filter implementation program actually consists of eleven parts, one for each programming form discussed in the previous section. Each programming form is analyzed using steady-state, statistical,

and upper bound techniques. The system weighting constants K_{ssj} , K_{stj} , and K_{ubj} are calculated using the equations of Table 1. K_{ssj} and K_{stj} were computed by both equations for debugging purposes; K_{ubj} was determined using the second equation. A weighted average of these constants was also used:

$$K_{waj} = \lambda_1 K_{ssj} + \lambda_2 K_{stj} + \lambda_3 K_{ubj} , \quad (7-1)$$

where

$$\lambda_1 + \lambda_2 + \lambda_3 = 1.$$

Therefore, a weighted average error can be calculated by

$$[e_o]_{wa} = \sum_{j=1}^S K_{waj} h'_j . \quad (7-2)$$

The weighting constants λ_i may be adjusted by the designer to emphasize the steady state, statistical, or upper bound errors.

The filter implementation program may be used in two modes, one for stored-program computers and one for special-purpose hardware; the two modes are distinguished by the manner in which the quantizer step lengths are chosen. In both the assumption is made that truncation

(or LSB-1) quantizers are used in the system. All errors must be halved by the user if roundoff quantizers are present.

In the stored program mode the maximum error h_j^1 of the j th quantizer is fixed by first simulating the ideal digital filter response to a "worst case" step input, which is an A/D input word of all "ones." During the transient response to this step, the maximum value of the filter output and internal variables is recorded. After the simulation has run a sufficient number of iterations for convergence, say 100, the maximum values are rounded up to the nearest power of two. Since the computer wordlength is a fixed number L_r , the quantizer intervals are found by

$$h_j^1 = \frac{\lceil |\text{Var}|_{\max} \rceil}{2^{L_r}} \text{ rounded-up} ; \quad (7-3)$$

h_1^1 is always assumed equal one.

In the special-purpose computer mode the register lengths are not fixed; therefore, a different method is used to find h_j^1 . The philosophy of this mode is to balance the effect of each quantizer in the system so that they all have relatively equal error contributions. This balancing is done by dividing equation (7-2) by K_{wa1} (with $h_1^1 = 1$):

$$\frac{[e_o]_{wa}}{K_{wa1}} = 1 + \sum_{j=2}^s \left(\frac{K_{waj}}{K_{wa1}} \right) h_j^1 \quad (7-4)$$

Each term in the summation is forced to be less than or equal one (to insure that the A/D will introduce an error as large or larger

than the other quantizers) by choosing

$$h_j' \leq \frac{K_{wal}}{K_{waj}} \quad (7-5)$$

A further restriction is that h_j' be a power of two; hence the ratio K_{wal}/K_{waj} is rounded down to the nearest power of two to find the actual h_j' to be used:

$$h_j' = [K_{wal}/K_{waj}] \text{ rounded down.} \quad (7-6)$$

Flow Charts

Fig. 37 demonstrates the flow of information in the main section of the filter implementation program. The input data to be given to the program is summarized below:

- 1) Transfer function coefficients in (6-1)
 a_0, a_1, a_2, b_1, b_2
- 2) Register lengths
 A/D, D/A, and wordlength of the stored-program computer
 or coefficient wordlength for the special-purpose computer.
- 3) Weighting coefficients in (7-1)
 $\lambda_1, \lambda_2, \lambda_3$

This is all the information needed to completely analyze the quantization errors for all the programming forms.

The first major task in the program is to find the poles and zeroes of the transfer function $D(z)$ and to set three flags which omit those programming forms which are unrealizable. The main program then calls a subprogram for each realizable programming form. Each called subprogram completely analyzes the quantization errors characteristic to that particular form and prints their detailed description. At the end of the program, final summaries of each program mode are listed for easy cross-reference.

F

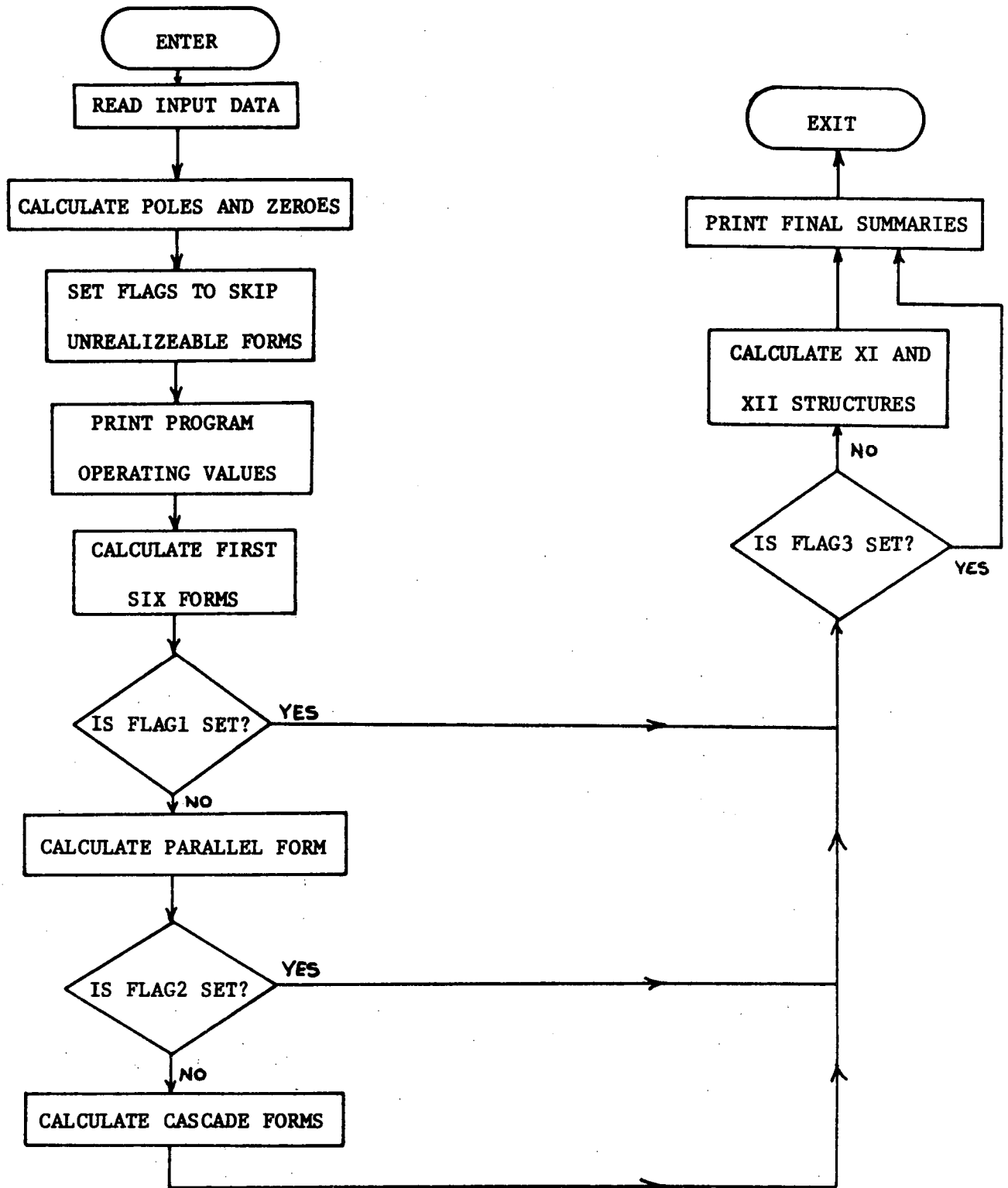


Fig. 37. Flow Chart of Main Program.

A general flow chart describing a subroutine for any given programming form is shown in Fig. 38. The first task is to calculate all of the parameters needed for the difference equations of the specified programming form; next, these parameters are quantized. The simulation difference equations are then calculated once for the step response and once for each quantizer in the system. During these simulations the system constants K_{ssj} , K_{stj} , and K_{ubj} are calculated. Finally the steady-state, statistical, and upper bound errors are calculated, as well as the weighted average error of equation (7-2), for both modes of program operation.

Source Listing

The filter implementation program consists of approximately 1800 source statements and is available in [49]. Also, a limited number of printed listings are available from Auburn University.

Summary

The filter implementation program has been developed using an IBM 360/50 using FORTRAN IV and OS360. In its final form the program takes approximately 3.5 minutes to compile and 25 seconds to load and execute. The execution time may be trimmed by limiting the simulation iterations to a smaller number, say 10 to 20.

Now the CAD program will be used to analyze two digital filters, one for each program operating mode.

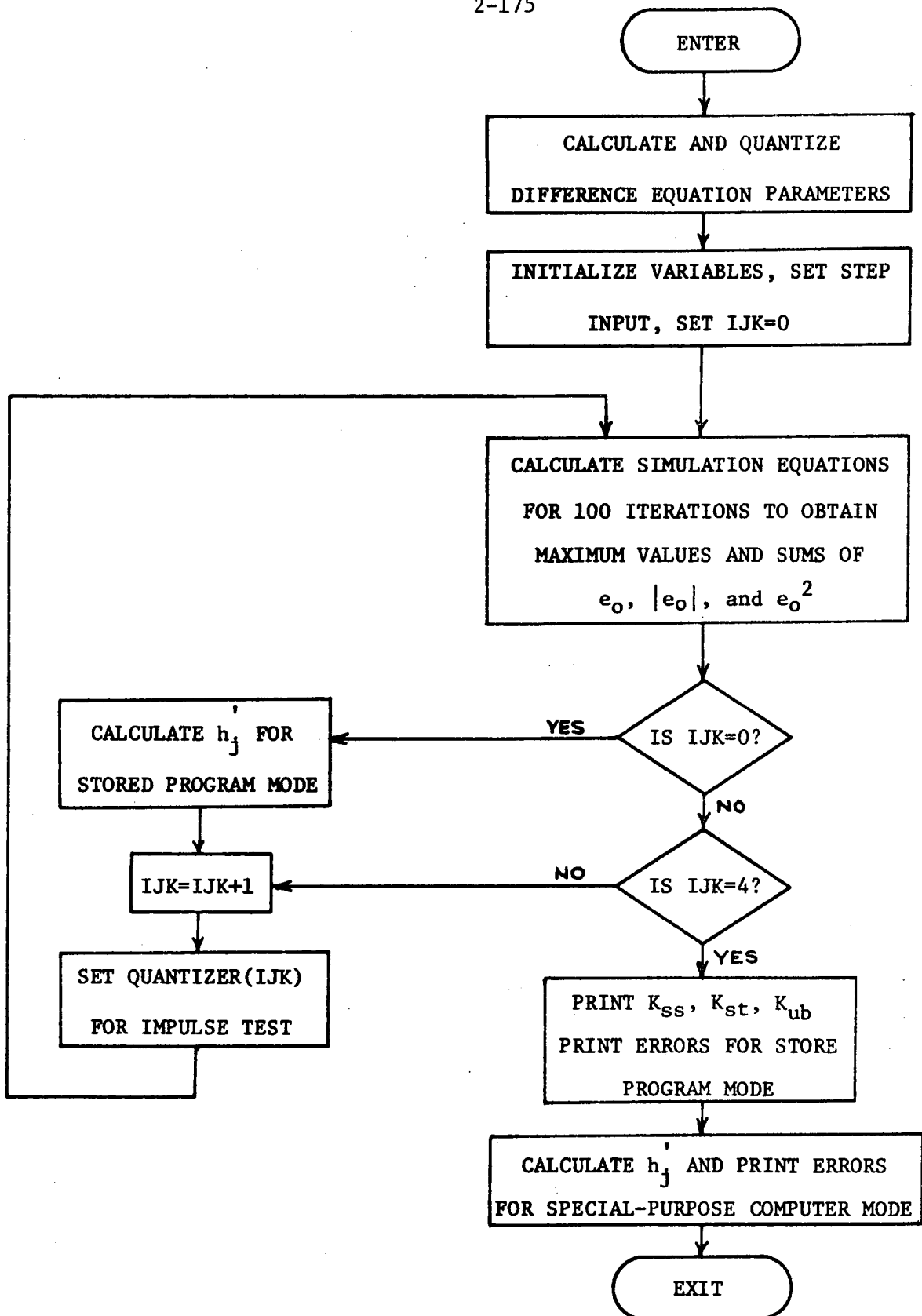


Fig. 38. Flow Chart for Programming Form.

Stored Program Mode

Consider the second order digital filter

$$D(z) = \frac{z^2 + .75z + 0.125}{z^2 + .50z + .0525} \quad (7-7)$$

Suppose that this filter is to be realized using a 16-bit minicomputer using a 11-bit A/D and 13-bit D/A as input-output equipment. The computer-aided design (CAD) program may be used in the stored-program mode of operation to aid the designer in programming the minicomputer. Table 3 is the final summary of quantization errors attributed to the filter above for its realizable programming forms. The $D(z)$ in (7-7) has real poles and zeroes; therefore, the $X1$ and $X2$ structures may not be used.

Using the weighted average errors in Table 3, the CAD program recommends that the filter in (7-7) be programmed by first the modified canonical form; and second, the parallel form. Note that all the programming forms give relatively good results; this is due to the fact that the internal quantizers and output quantizers contribute only a minor part of the total quantizing error. The A/D and D/A wordlengths chosen in this example are responsible for these results.

TABLE 3. Final Summary: Stored-Program Computer Mode

Programming Form	Steady-State Error	RMS Error	Maximum Error	Weighted Average
Direct	1.730	.904	1.872	1.502
Modified Direct	1.770	.945	1.985	1.567
Standard	1.758	.923	.912	1.531
Modified Standard	1.810	.986	2.098	1.632
Canonical	1.745	.903	1.863	1.504
Modified Canonical	1.714	.889	1.831	1.478
Parallel	1.725	.898	1.850	1.491
Cascade	1.780	.922	1.901	1.534
Modified Cascade	1.813	.947	1.959	1.573

Special-Purpose Computer Mode

Suppose that a special-purpose computer is to be constructed to implement the following z-domain transfer function:

$$D(z) = \frac{z^2 - 1.862z + .895}{z^2 - .2500} \quad (7-8)$$

Again, if an 11-bit A/D is to be used, the CAD program gives the results shown in Table 4. From the table, the weighted average error suggests that the direct form is best; the modified canonical form, second.

Direct form. The program prints out an analysis of each programming form which may be used for (7-8). See Table 5. The system error weighting constants (K_{ss} , K_{st} , and K_{ub}) are summarized as well as the λ 's of (7-1), the maximum quantizing error h' of each quantizer, and the form factor. The form factor is interpreted as follows

$$\text{FORM} = I, J,$$

where

I = total number of bits for the register

J = number of bits to the right of the binary point.

A negative J indicates the least significant bit has a value greater than one. From Table 5, $h'_2 = 2h'_1$ and $h'_3 = 4h'_1$. The CAD program always

TABLE 4. Final Summary: Special-Purpose Computer Mode

Programming Form	Steady-State Error	RMS Error	Maximum Error	Weighted Average
Direct	3.377	3.177	8.343	4.966
Modified Direct	4.711	3.773	9.676	6.053
Standard	4.711	3.773	9.676	6.053
Modified Standard	6.044	4.369	11.009	7.141
Canonical	2.088	4.006	12.019	6.038
Modified Canonical	3.000	3.884	11.019	5.968
Parallel	4.855	4.158	11.742	6.918

TABLE 5: The Direct Printout

STEADY-STATE ANALYSIS

KSS(1) = 0.044

KSS(2) = 1.000

KSS(3) = 0.333

STATISTICAL ANALYSIS

KST(1) = 1.426

KST(2) = 0.577

KST(3) = 0.149

ERROR BOUND ANALYSIS

KUB(1) = 5.009

KUB(2) = 1.000

KUB(3) = 0.333

SPECIAL-PURPOSE COMPUTER MODE

LAMBDA(1) = 0.333 H(1) = 1.0 FORM = 11,0

LAMBDA(2) = 0.333 H(2) = 2.0 FORM = 10,-1

LAMBDA(3) = 0.333 H(3) = 4.0 FORM = 9,-2

STEADY-STATE ERROR = 3.377

PERCENT Q1 = 1.3

PERCENT Q2 = 59.2

PERCENT Q3 = 39.5

RMS ERROR = 3.177

PERCENT Q1 = 44.9

PERCENT Q2 = 36.3

PERCENT Q3 = 18.8

MAXIMUM ERROR BOUND = 8.343

PERCENT Q1 = 60.0

PERCENT Q2 = 24.0

PERCENT Q3 = 16.0

WEIGHTED AVERAGE ERROR = 4.966

PERCENT Q1 = 43.5

PERCENT Q2 = 34.6

PERCENT Q3 = 21.9

assumes $h_1' = 1$. Also, the form factor of Q_2 , the output quantizer, suggests that a 10-bit D/A may be used.

Modified canonical form. The CAD program output for the modified canonical form is shown in Table 6. Note that $h_2' = 2h_1'$ and $h_3' = h_1'$ for this programming form.

Closed-Loop Comparison

The second-order digital filter in (7-8) has been analyzed in [53] for a closed-loop sampled-data control system. The block diagram for the control loop is shown in Fig. 39. Statistical and upper bound techniques were employed to design the compensator of the control loop for both the direct and modified canonical forms; system simulations were employed to verify the results. Table 7 presents a comparison of the open-loop results of this paper and the closed-loop results of [53]. Note that they agree very closely.

One observation should be made at this point. The register lengths determined by the open-loop design procedures of this paper are in general larger than those required in closed-loop applications. Stable feedback systems generally tend to reduce the maximum values of the digital filter variables and thus the number of bits needed to represent these variables in the special-purpose computer.

TABLE 6: The Modified Canonical Printout

STEADY-STATE ANALYSIS

KSS(1) = 0.044
 KSS(2) = 1.000
 KSS(3) = -0.956

STATISTICAL ANALYSIS

KST(1) = 1.426
 KST(2) = 0.577
 KST(3) = 1.303

ERROR BOUND ANALYSIS

KUB(1) = 5.009
 KUB(2) = 1.000
 KUB(3) = 4.009

SPECIAL-PURPOSE COMPUTER MODE

LAMBDA(1) = 0.333	H(1) = 1.0	FORM = 11,0
LAMBDA(2) = 0.333	H(2) = 2.0	FORM = 10,-1
LAMBDA(3) = 0.333	H(3) = 1.0	FORM = 12,0

STEADY-STATE ERROR = 3.000

PERCENT Q1 = 1.4
 PERCENT Q2 = 66.7
 PERCENT Q3 = 31.9

RMS ERROR = 3.884

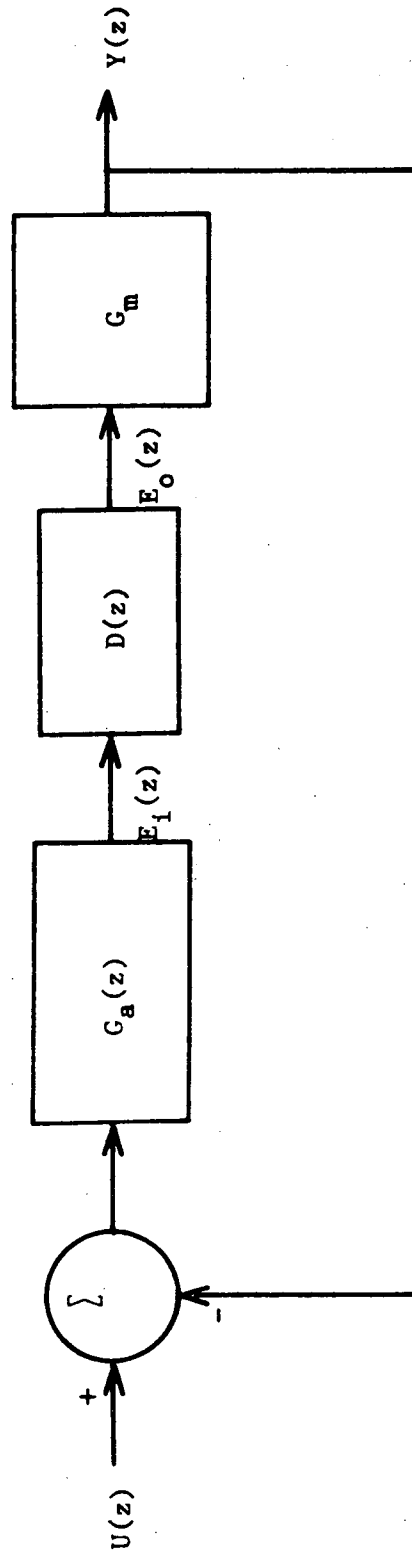
PERCENT Q1 = 36.7
 PERCENT Q2 = 29.7
 PERCENT Q3 = 33.6

MAXIMUM ERROR BOUND = 11.019

PERCENT Q1 = 45.5
 PERCENT Q2 = 18.1
 PERCENT Q3 = 36.4

WEIGHTED AVERAGE ERROR = 5.968

PERCENT Q1 = 36.1
 PERCENT Q2 = 28.8
 PERCENT Q3 = 35.1



$$G_a(z) = 3.17 \times 10^{-9} \frac{z^2 + Ez + 1}{(z-1)(z^2 + Fz + 1)}$$

$$E = 3.99$$

$$F = -1.97$$

$$T = .001$$

$$G_m = 2.52 \times 10^7$$

$$D(z) = \frac{z^2 - \frac{1907}{1024} + \frac{917}{1024}}{z^2 - \frac{256}{1024}}$$

Fig. 39. Control-loop [53].

TABLE 7: Open-Loop Versus Closed-Loop

Programming Form	Open-Loop Results	Closed-Loop Results [17]
Direct	$h'_2 = 2h'_1$	$h'_2 = h'_1$
	$h'_3 = 4h'_1$	$h'_3 = 4h'_1$
Modified	$h'_2 = 2h'_1$	$h'_2 = h'_1$
Canonical	$h'_3 = h'_1$	$h'_3 = .5h'_1$

Conclusion

This section has presented a computer-aided design technique useful in implementing digital filters expressed as z-domain transfer functions. Two examples have been given to illustrate the stored-program and special-purpose modes of operation of the CAD program. Also, the program, which analyzes the filter's "open-loop" quantization errors, gives results closely matching a "closed-loop" design. This CAD program should be used as a tool for obtaining a "first guess" at the best way to program a digital filter. If a closed-loop simulation is available for the system in which the digital filter will be used, then the CAD program design may be adjusted to give better loop performance.

Although the program as presented has been designed for second-order modules, it can be used as a subroutine in larger programs to match pole-zero pairs for higher order realizations, or to indicate the proper cascade ordering of second-order modules. The CAD program may be a powerful tool to the digital filter (or controller) designer if its results are properly interpreted.

VIII. APPLICATIONS OF DIGITAL FILTERING

Digital Filtering has found many diverse applications in recent years. This section lists several of them and points the interested reader to the open literature for detailed descriptions.

The following list presents typical applications for digital filters:

1. Sampled-Data Control Systems
 - a. General [38, 77]
 - b. Pendulous Integrating Gyroscopic Accelerometer [78, 79].
 - c. Saturn V Thrust Vector Control [80, 81]
2. Speech Processing
 - a. General [82]
 - b. Vocoder [83]
 - c. Equalizers [84]
3. Radar and Sonar Signal Processing
 - a. General [85, 86]
 - b. MTI Filters [87, 88]
 - c. Tracking Filters [89, 90]
4. Spectral Analysis and Synthesis
 - a. Narrow Band Filters
 - b. FFT [91]
 - c. Frequency Synthesis [92]
5. Vibrations and Acoustic Testing [93]
6. Image Processing
 - a. General [24, 94, 95]
 - b. Image Enhancement [94, 95, 96]
 - c. Pattern Recognition [97]
7. Seismic Processing [7, 9]
8. Biomedical Processing [94, 95, 97, 98]
9. Synthesis of Speech and Music [99]

Many other applications of digital filtering are also important with the number of new ones ever increasing.

REFERENCES

- [1] R. R. Read and C. S. Burrus, "Use of the Geometry of Partial Sums in Digital Filter Analysis," IEEEETAU, Vol. AU-20, Aug. 72, pp. 213-218.
- [2] T. G. Stockham, Jr., Chapter 7, Digital Processing of Signals (B. Gold and C. M. Rader), New York: McGraw-Hill, 1969.
- [3] L. R. Rabiner and R. W. Schafer, "Recursive and Non-Recursive Realization of Digital Filter Designed by Frequency Sampling Techniques," IEEEETAU, Vol. AU-19, September 71, pp. 200-207.
- [4] T. S. Huang, "Digital Signal Processing - Applications to Speech and Image Processing," Course Notes, UCLA, July, 1972.
- [5] "General Principles of Digital Filtering and a Survey of Filters in Current Range Use," IRIG Document 122-71, Data Reduction and Computing Group, Range Commanders Council, U.S. Air Force, December, 1971.
- [6] W. C. Kellogg, "Time Domain Design of Nonrecursive Least Mean-Square Digital Filters," IEEEETAU, Vol. AU-20, June, 1972, pp. 155-158.
- [7] E. A. Robinson and S. Treitel, "Principles of Digital Wiener Filtering," Geophysical Prospecting, Vol. 15, September, 1967, pp. 311-333.
- [8] J. D. Markel, "Digital Inverse Filtering - A New Tool for Formant Trajectory Estimation," IEEEETAU, Vol. AU-20, June, 1972, pp. 129-137.
- [9] K. L. Peacock and S. Treitel, "Predictive Deconvolution: Theory and Practice," Geophysics, Vol. 34, April, 1969, pp. 155-169.
- [10] H. B. Voelcker and E. E. Hartquist, "Digital Filtering via Block Recursion," IEEEETAU, Vol. AU-18, June, 1970, pp. 169-176.
- [11] H. O. Helms, "Fast Fourier Transform Method of Computing Difference Equations and Simulating Filters," IEEEETAU, Vol. AU-15, June, 1967, pp. 85-90.

- [12] D. Chanoux, "Synthesis of Recursive Digital Filters Using the FFT," IEEEETAU, Vol. AU-18, June, 1970, pp. 211-212.
- [13] R. Reed and J. Meek, "Digital Filters with Poles Via the FFT," IEEEETAU, Vol. AU-19, December, 1971, pp. 322-323.
- [14] J. P. Thiran, "Recursive Digital Filters with Maximally Flat Group Delay," IEEE Trans. on Circ. Theory, Vol. CT-18, November, 1971, pp. 659-664.
- [15] T. H. Crystal and L. Ehrman, "The Design and Application of Digital Filters with Complex Coefficients," IEEEETAU, Vol. AU-16, September, 1968, pp. 315-320.
- [16] E. P. F. Kan and J. K. Aggarwal, "Randomly Sampled Digital Filters," IEEEETAU, Vol. AU-20, March, 1972, pp. 52-57.
- [17] E. P. F. Kan and J. K. Aggarwal, "Multirate Digital Filtering," IEEEETAU, Vol. AU-20, August, 1972, pp. 223-224 (Correspondence).
- [18] T. S. Huang, "Stability of Two-Dimensional Recursive Filters," IEEEETAU, Vol. AU-20, June, 1972, pp. 158-163.
- [19] T. S. Huang, "Two-Dimensional Windows," IEEEETAU, AU-20, March, 1972, pp. 88-89.
- [20] J. G. Proakis, "Adaptive Digital Filters for Equalization of Telephone Channels," IEEEETAU, Vol. AU-18, June, 1970, pp. 195-200.
- [21] A. R. M. Noton, Introduction to Variational Methods in Control Engineering, New York, Pergamon Press, 1965.
- [22] G. Williamson, "Optimal Controllers for Homing Missiles," RE-TR-68-15, U. S. Army Missile Command, Redstone Arsenal, AL, September, 1968.
- [23] R. E. Kalman and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," J. Basic Eng., March, 1961, pp. 95-108.
- [24] A. B. Oppenheim, R. W. Schafer, and T. G. Stockham, Jr., "Nonlinear Filtering of Multiplied and Convolved Signals," IEEEETAU, Vol. AU-16, September, 1968, pp. 437-566.
- [25] J. R. Heath and C. C. Carroll, "Special-Purpose Computer Organization for Double-Precision Realization of Digital Filters," IEEEETC, Vol. C-19, December, 1970, pp. 1146-1152.

- [26] C. C. Carroll and J. W. Jones, "A Special-Purpose Computer Realization of a Time-Shared Digital Filter," Technical Report Number 10, NAS8-20163, Engineering Experiment Station, Auburn, Alabama, August, 1968.
- [27] A. V. Oppenheim, "Realization of Digital Filters Using Block-Floating-Point Arithmetic," IEEETAU, Vol. AU-18, June, 1970, pp. 130-136.
- [28] A. W. Crooke and J. W. Craig, "Digital Filters for Sample-Rate Reduction," IEEETAU, Vol. AU-20, October, 1972, pp. 308-315.
- [29] A. A. G. Requicha and H. B. Voelcker, "Design of Nonrecursive Filters by Specification of Frequency-Domain Zeroes," IEEETAU, Vol. AU-18, December, 1970, pp. 464-470.
- [30] L. R. Rabiner, B. Gold, and C. A. McGonegal, "An Approach to the Approximation Problem for Nonrecursive Digital Filters," IEEETAU, Vol. AU-18, June, 1970, pp. 83-106.
- [31] B. Gold and K. L. Jordan, Jr., "A Direct Search Procedure for Designing Finite Duration Impulse Response Filters," IEEETAU, Vol. AU-17, March, 1969, pp. 33-36.
- [32] T. J. McCreary, "On Frequency Sampling Filters," IEEETAU, Vol. AU-20, August, 1972, pp. 222-223.
- [33] H. D. Helms, "Nonrecursive Digital Filters: Design Methods for Achieving Specifications on Frequency Response," IEEETAU, Vol. AU-16, September, 1968, pp. 336-342.
- [34] H. D. Helms, "Digital Filters with Equiripple or Minimax Response," IEEETAU, Vol. AU-19, March, 1971, pp. 87-93.
- [35] R. W. Hankins, "Design Procedure for Equiripple Nonrecursive Digital Filters," Technical Report 485, MIT Research Laboratory of Electronics, Cambridge, MA, May 12, 1972.
- [36] R. K. Ontes, "An Elementary Design Procedure for Digital Filters," IEEETAU, Vol. AU-16, September, 1968, pp. 330-335.
- [37] R. M. Golden, "Digital Filter Synthesis by Sampled-Data Transformation," IEEETAU, Vol. AU-16, September, 1968, pp. 321-329.
- [38] B. C. Kuo, Analysis and Synthesis of Sampled-Data Control Systems. Englewood Cliffs, NJ, Prentice-Hall, Inc., 1963.

- [39] R. Fletcher, Optimization. New York: Academic Press, 1969.
- [40] T. C. Hsia, "On Synthesis of Optimal Digital Filters," Proc. First Asilomar Conference on Circuits and Systems, November, 1967.
- [41] D. B. Kimsey and H. T. Nagle, "Digital Filter Implementation by Minicomputer," Proc. IEEE Region 3 Convention, April 10-12, 1972, pp. C3-1, C3-4.
- [42] L. R. Rabiner and K. Steiglitz, "The Design of Wide-band Recursive and Non-Recursive Digital Differentiators," IEEE TAU, Vol. AU-18, June, 1970, pp. 204-209.
- [43] D. W. Tufts and J. T. Francis, "Designing Digital Low-Pass Filters - Comparison of Some Methods and Criteria," IEEE TAU, Vol. AU-18, December, 1970, pp. 487-494.
- [44] S. C. D. Roy, "On Maximally Flat Sharp Cutoff Low-Pass Filters," IEEE TAU, Vol. AU-19, March, 1971, pp. 58-63.
- [45] M. C. Agarwal and A. S. Sedra, "On Designing Sharp Cutoff Low-Pass Filters," IEEE TAU, Vol. AU-20, June, 1972, pp. 138-141.
- [46] A. V. Oppenheim, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," Proceedings of the IEEE, August, 1972, pp. 957-976.
- [47] R. K. Ontes and L. P. McNamee, "Instability Thresholds in Digital Filters Due to Coefficient Rounding," IEEE TAU, Vol. AU-18, December, 1970, pp. 456-463.
- [48] E. Avenhaus, "On the Design of Digital Filters with Coefficients of Limited Wordlength," IEEE TAU, August, 1972, pp. 206-212.
- [49] H. T. Nagle, Jr., and M. M. Edgeworth, "Computer Aided Design of Digital Filters," TR#14, NAS8-20163, George C. Marshall Space Flight Center, Huntsville, AL, September, 1971.
- [50] J. B. Slaughter, "Quantization Errors in Digital Control Systems," IEEE TAC, Vol. AC-19, January, 1964, pp. 70-74.
- [51] B. Widrow, "Statistical Analysis of Amplitude Quantized Sampled-Data Systems," AIEE Trans. on Appl. and Ind., No. 52, January, 1961.
- [52] H. T. Nagle, Jr., "Comments on 'A Least Upper Bound on Quantization Error'," IEEE TAC, Vol. AC-14, August, 1969, pp. 433-434.

- [53] H. T. Nagle, Jr., and C. C. Carroll, "Memory Sizing for Digital Filters," Proc. of the IFIP Congress '71, Ljubljana, Yugoslavia, August 23-25, 1971, pp. TA-4-129, 133.
- [54] E. P. F. Kan and J. K. Aggarwal, "Minimum - Deadband Design of Digital Filters," IEEETAU, Vol. AU-19, December, 1971, pp. 292-296.
- [55] S. R. Parker and S. F. Hess, "Limit-Cycle Oscillations in Digital Filters," IEEETCT, Vol. CT-18, November, 1971, pp. 687-697.
- [56] P. M. Ebert, J. E. Mazo, and M. C. Taylor, "Overflow Oscillations in Digital Filters," BSTJ, Vol. 48, 1969, pp. 2999-3020.
- [57] H. T. Nagle, Jr., and C. C. Carroll, "Organizing a Special-Purpose Computer to Realize Digital Filters for Sampled-Data Systems," IEEETAU, Vol. AU-16, September, 1968, pp. 398-412.
- [58] I. W. Sandberg, "Floating-point Roundoff Accumulation in Digital Filter Realization," BSTJ, Vol. 46, October, 1967, pp. 1774-1791.
- [59] B. Liu and T. Kaneko, "Error Analysis of Digital Filters Realized with Floating-Point Arithmetic," Proc. IEEE, Vol. 57, October, 1969, pp. 1735-1747.
- [60] B. Liu, "Effect of Finite Wordlength on the Accuracy of Digital Filters - A Review," IEEETCT, Vol. CT-18, November, 1971, pp. 670-677.
- [61] E. P. F. Kan and J. K. Aggarwal, "Error Analysis of Digital Filter Employing Floating-Point Arithmetic," IEEETCT, Vol. CT-18, November, 1971, pp. 678-686.
- [62] J. F. Kaizer, "Some Practical Considerations in the Realization of Linear Digital Filters," Proc. 3rd Annual Allerton Conf. on Circuit and System Theory, 1965, pp. 621-633.
- [63] J. W. Henderson, Jr., "A Comparison of Different Realizations of a Second Order Digital Filter with Regard to Quantization Errors," Masters Thesis, Auburn University, Auburn, AL, June 9, 1970.
- [64] R. E. Crochiere, "Digital Ladder Structures and Coefficient Sensitivity," IEEETAU, Vol. AU-20, October, 1972, pp. 240-246.

- [65] A. E. Vereshkin, et.al., "Two New Structures for the Implementation of a Discrete Transfer Function with Complex Poles," Automation and Remote Control, September, 1968, pp. 1416-22.
- [66] P. M. DeRusso, R. J. Roy and C. M. Close, State Variables for Engineers, New York, N. Y., John Wiley and Sons, Inc., 1965.
- [67] S. K. Mitra and R. J. Sherwood, "Canonic Realizations of Digital Filters Using the Continued Fraction Expansion," IEEEETAU, Vol. AU-20, August, 1972, pp. 185-194.
- [68] A. Antonion, "Realization of Digital Filters," IEEEETAU, Vol. AU-20, March, 1972, pp. 95-97.
- [69] C. S. Burrus, "Block Realization of Digital Filters," IEEEETAU, Vol. AU-20, October, 1972, pp. 230-235.
- [70] L. B. Jackson, "Roundoff Noise Analysis for Fixed-Point Digital Filters Realized in Cascade or Parallel Form," IEEEETAU, Vol. AU-18, June, 1970, pp. 107-122.
- [71] R. M. Golden, "A Computer Program for the Design of Continuous and Digital Transfer Functions," Autonetics Publication TM68-572-21-8, August, 1968.
- [72] T. W. Parks and J. H. McClellan, "A Program for the Design of Linear Phase Finite Impulse Response Digital Filters," IEEEETAU, Vol. AU-20, August, 1972, pp. 195-199.
- [73] L. R. Rabiner, "Linear Program Design of Finite Impulse Response (FIR) Digital Filters," IEEEETAU, Vol. AU-20, October, 1972, pp. 280-288.
- [74] P. N. Robinson and G. S. Robinson, "A Computer Method for Obtaining z-Transforms," IEEEETAU, Vol. AU-20, March, 1972, pp. 98-99.
- [75] K. Steiglitz, "Computer-Aided Design of Recursive Digital Filters," IEEEETAU, Vol. AU-18, June, 1970, pp. 123-129.
- [76] A. Fellweis, "Some Principles of Designing Digital Filters Imitating Classical Filter Structures," IEEEETCT, Vol. CT-18, March, 1971, pp. 314-316.
- [77] J. A. Cadyow and H. R. Martens, Discrete-Time and Computer Control Systems. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1970.

- [78] C. C. Carroll, et.al., "The Hybrid Realization of a Digital Controller for the PIGA Control Loop," TR#6, NAS8-20163, George C. Marshall Space Flight Center, Huntsville, AL, September, 1967.
- [79] R. White, H. T. Nagle, and C. C. Carroll, "Organization of a High-Speed Stored-Program Special-Purpose Computer for the Realization of Digital Filters," TR#13, NAS8-20163, George C. Marshall Space Flight Center, NASA, Huntsville, AL, May, 1971.
- [80] C. L. Phillips, et.al., "Digital Compensation of the Thrust Vector Control System," TR#8, NAS8-11274, George C. Marshall Space Flight Center, NASA, Huntsville, AL, May, 1967.
- [81] H. T. Nagle, Jr., and C. C. Carroll, "A Special-Purpose Realization of a Third-Order Digital Filter for the PIGA Control Loop," TR#9, NAS8-20163, George C. Marshall Space Flight Center, NASA, Huntsville, AL, May, 1968.
- [82] R. W. Schafer, "A Survey of Digital Speech Processing Techniques," IEEETAU, Vol. AU-20, March, 1972, pp. 28-35.
- [83] L. K. Schweizer, "Problems in Realizing a Digital Vocoder and Novel Solutions," IEEETAU, Vol. AU-19, March, 1971, pp. 94-96.
- [84] F. Eggimann, "Computer Simulation of an Automatic Adaptive Equalizer for Real Telephone Channels and Free Data Format," IEEETAU, Vol. AU-18, December, 1970, pp. 434-438.
- [85] K. V. Schlachta, "Digital Radar Recording and Analysis," IEEETAU, Vol. AU-18, December, 1970, pp. 399-403.
- [86] J. D. Echard and R. R. Boorstyn, "Digital Filtering for Radar Signal Processing Applications," IEEETAU, Vol. AU-20, March, 1972, pp. 42-52.
- [87] A. I. Zverev, "Digital MIT Radar Filters," IEEETAU, Vol. AU-16, September, 1968, pp. 422-432.
- [88] R. Roecker, "The Application of Digital Filters for Moving Target Indication," IEEETAU, Vol. AU-19, March, 1971, pp. 72-77.
- [89] A. J. Monroe, Digital Processes for Sampled-Data Systems. New York: John Wiley and sons, Inc., 1962
- [90] N. Morrison, Introduction to Sequential Smoothing and Prediction. New York: McGraw-Hill, 1969.

- [91] S. Bertram, "Frequency Analysis Using the Discrete Fourier Transform," IEEE TAU, Vol. AU-18, December, 1970, pp. 495-500.
- [92] J. Tierney, C. M. Rader, and B. Gold, "A Digital Frequency Synthesizer," IEEE TAU, Vol. AU-19, March, 1971, pp. 48-57.
- [93] A. G. Ratz, "Statistical Effects in Automatic Random Equalizer," IEEE TIM, Vol. IM-16, December, 1967.
- [94] Special Issue on Digital Picture Processing, Proceedings of the IEEE, Vol. 60, No. 7, July, 1972.
- [95] Special Issue on Two-Dimensional Digital Signal Processing, IEEE TC, Vol. C-21, Number 7, July, 1972.
- [96] H. C. Andrews, A. G. Tescher, and R. P. Kreiger, "Image Processing by Digital Computer," IEEE Spectrum, Vol. 9, July, 1972, pp. 20-32.
- [97] Special Issue on Digital Pattern Recognition, Proceedings of the IEEE, Vol. 60, No. 10, October, 1972.
- [98] G. Dumermuth, et.al., "Numerical Analysis of Electroencephalographic Data," IEEE TAU, Vol. AU-18, December, 1970, pp. 404-411.
- [99] L. L. Beranek, "Digital Synthesis of Speech and Music," IEEE TAU, Vol. AU-18, December, 1970, pp. 426-433.

PART THREE

MECHANIZATION
OF DIGITAL
FILTERS

PART THREE: MECHANIZATION OF DIGITAL FILTERS

TABLE OF CONTENTS

I. Introduction	3-1
II. General Purpose Computer Implementations	3-4
A. Simulation	3-4
B. Real Time Programming.	3-9
III. Minicomputer Implementations	3-11
A. Hardware Requirements.	3-11
B. Operating System	3-16
C. Assembly Programs.	3-19
D. Experimental Results	3-22
IV. Special Purpose Computers.	3-25
A. Implementation by Sample and Hold Devices with Analog Networks.	3-25
B. Hybrid Implementation.	3-30
C. Digital Implementation	3-41
1. Input/Output Components.	3-45
2. Arithmetic Unit.	3-49
3. Memory Design.	3-63
4. Controller Design.	3-67
D. Implementation by Microprogrammable SP Computer.	3-70
1. Input/Output Unit.	3-76
2. Arithmetic Unit.	3-77
3. Memory	3-82
4. Control Unit	3-93

E. Time-Sharing of a Digital Filter Implementation.	3-98
F. Range Switching Digital Filter Implementation.	3-105
G. LSI Digital Filter Implementation.	3-112
H. Commercial Digital Filters	3-113
V. FFT Hardware	3-117
A. Commercial Equipment	3-117
B. MIT Fast Digital Processor	3-118
REFERENCES	3-125

I. INTRODUCTION

In recent years a trend has been developing to replace analog systems with digital systems. This rate of replacement has been directly related to the technological advances in the manufacturing of digital logic building blocks. With the advent of large-scale-integration, a particular class of digital networks, called digital filters, has become economically practical in such areas as stabilization of control systems, spectrum analysis, voice and speech analysis, radar, medical electronics and virtually any other analog filter function [1,2].

Digital filtering has been defined in PARTS ONE and TWO as a computational process consisting of digital multiplications, additions and delays whereby one sequence of numbers is transformed into another sequence. This transformation may be specified by a transfer function in the z -domain, $D(z)$, or by a set of linear difference equations with constant coefficients. Assuming knowledge of these coefficients, digital filter realization procedures [1,3,4,5] consist of the design of a digital system to solve these difference equations. The difference equations may be solved with a software program and a general purpose computer or with the use of a special-purpose (SP) computer [6,7,8,9, 10,11,12], a technique which has become increasingly popular.

In the SP computer realizations, a particular digital filter programming form is selected and the computer is designed accordingly [13,14,15,16,17]. Particular attention must be given to assure that the hardware organization meets the system specifications for coefficient quantization, signal amplitude quantization, and quantization noise levels introduced into the system by the digital filter implementation. At the present time no systematic design procedure has been developed to accomplish these goals. Typically one designer specifies the digital filter coefficients and another specifies a hardware implementation.

The state-of-the-art in digital filter implementation is represented in [1,3,7,8,9,10,12,18,19,20,21]. Perhaps the most interesting are the IC model in [1] and the programmable design of [12]. The IC model is available from Autonetics Division of North American Rockwell. A digital filter implemented with this technology is small and can realize third-order filters at sampling rates of up to 5kHz. However, poles must be real and the parallel programming form is the only one available. The commercial units also have restricted programming forms, or implementation is done by frequency transformations which limit their use to applications in which minimum time delay and high speed sampling are not specified. It has been shown in [14] that some of the programming forms have different characteristics, and it is desirable in many cases to be able to select the programming form. The need for a selectable programming form along with the desirable features of LSI implementation offer a challenge to the system designer.

Add the necessity for real-time fault diagnosis and standardized CAD procedures to the list and the system design goals are complete.

Now that the theory of digital filtering has been presented in PART TWO, we will examine four mechanization techniques for digital filters. The four techniques are 1) general-purpose computers, 2) mini-computers, 3) special-purpose computers, and 4) FFT hardware. A discussion of all techniques will be presented starting with mechanization (implementation) by general-purpose (GP) computers.

C4

II. GENERAL PURPOSE COMPUTER IMPLEMENTATIONS

Of the four implementation techniques, the GP computer is the least attractive, with the main reason being that most GP computers possess excessive computing capabilities to be used only for difference equation calculations. If this were done, there would be large portions of the computer hardware that would never be used thereby making this type implementation overly expensive.

GP computers do have a useful application in digital filter implementation in that they may be used to simulate other implementation designs (an example being by special-purpose computers) or for real-time programming of a GP computer to implement a digital filter as well as other computational chores. Let us now look at these two aspects of using a GP computer in the design of a digital filter system.

Simulation

The most common implementation of a digital filter is by special-purpose computer. When designing a special-purpose computer for the implementation of the filter in a particular programming form, one of

the first steps that must be done is deciding on word length requirements for the input word, output word and internal variable ($m(kT - T)$, $m(kT - 2T)$, etc. of the difference eqs.) wordlengths and possibly arithmetic schemes. This can be accomplished by techniques such as the CAD program presented earlier. Once a design is recommended, it is good engineering practice to simulate the system on a GP computer to verify all the design parameters. With most higher level languages, logical programming may be done such that every aspect of the design may be simulated. If this approach is taken the system designer may "change something" and observe its effects; this technique may be used to "optimize" the final system design.

As an example of a digital filter implementation simulation, the program below was written in FORTRAN and run on an IBM 360 digital computer to simulate the "range-switching" filter described in [8] employed in a nulling type control loop. The program was written so that the "range-switching" effects on the output of the loop could be observed and the effect the wordlengths had on the output response for a particular input, which in this case is a sine wave of specified amplitude and frequency.

FORTRAN SOURCE PROGRAM FOR SIMULATION OF PIGA LOOP
WITH A NOISE INPUT

SOURCE DECK

```

C      TIME DOMAIN SIMULATION OF THE COMPENSATED SYSTEM
      DIMENSION X1(2),X2(2),X3(2),      D(2)
      COMMON/COM2/RM(1001),ITER,TES
      COMMON/COM1/XP(3,2),BQ(1),C(2)
      1 FORMAT(1H ,13,2X,1P9E13.4)
      CALL INPUT
      2 FORMAT(1H1)
      N=1
      H=1.0E+05
      WN=184.0
      T=0.001
      GP=56.2
      GT=321000.0
C      GDA IS THE TOTAL LOOP GAIN
      GDA=0.083
      W1=SIN(WN*T)/WN
      W2=COS(WN*T)
      W3=(1.0-W2)/WN**2
      W4=(T-W1)/H
      W5=(1.0-W2)/H
      W6=WN*SIN(WN*T)
      W7=W6/H
      GDIG=GAD/GT/GP/(T-W1)*H
      WRITE(6,10) GDIG
      10 FORMAT(1H0,7HKDIG = ,1PE11.4)
      TEST=+2000.0*980.0/4.0
      DO 669 NAGL=1,2
669      XP(NAGL,1)=0.0
      8 BDC=0.0
      BQ(1)=0.0
      X1(1)=0.0
      X2(1)=0.0
      X3(1)=0.0
      D(1)=0.0
      C(1)=0.0
      COFS=0.0
      DO 5 I=1,ITER
      R=TES*980.*RM(I)
4      WRITE(6,1) I,BDC,BQ(N),X1(N),XP(1,1),D(N),R,COFS,C(1)
C      BEGIN ANALOG PORTION SIMULATION
      X1(N+1)=X1(N)+W1*X2(N)+W3*X3(N)+W4*(R-GR*D(N))
      X2(N+1)=W2*X2(N)+W1*X3(N)+W5*(R-GR*D(N))
      X3(N+1)=W2*X3(N)+W6*X2(N)+W7*(R-ST*D(N))
      BDC=GP*X1(N+1)

```

```

C      END    ANALOG PORTION SIMULATION
C      BEGIN DIGITAL UNIT SIMULATION
      UI=BDC
      CALL DIGCOM (UI,YP)
      D(N+1)=GDIG*YP
C      END    DIGITAL UNIT SIMULATION
      COFS=GT*D(N+1)
      X1(N)=X1(N+1)
      X2(N)=X2(N+1)
      X3(N)=X3(N+1)
5 D(N)=D(N+1)
      STOP
      END

      SUBROUTINE DIGCOM(UI,YP)
      COMMON/COM1/SP(3,2),BQ(1),C(2)
      AO=1.0
      A1=-119./64.
      A2=57./64.
      B1=0.0
      B2=0.0
      FX=256.
      UI=UI/3.0*FX
      EX=1.0
      CALL ROUND (UI,EX,FX)
      UP=UI
Z      BQ(1)=UP*3.0/FX
      UI=UI*3.0/FX
      IF(ABS(UP)-16.0) 2,3,3
2      C(2)=0.0
      GO TO 4
3      C(2)=1.0
      UP=UP/16.
      IUP=UP
      UP=IUP
      FX=16.
4      IF(C(2)-C(1)) 4,6,7
5      XP(1,1)=16.*XP(1,1)
      XP(2,1)=16.*XP(2,1)
      AX=4.0
      BX=63.75
      CALL ROUND(XP(1,1),AX,BX)
      CALL ROUND(XP(2,1),AX,BX)
      GO TO 6
7      XP(1,1)=XP(1,1)/16.0
      XP(2,1)=XP(2,1)/16.
      AX=4.0
      BX=63.75
      CALL ROUND (XP(1,1),AX,BX)
      CALL ROUND (XP(2,1),AX,BX)
6      SP(1,2)=-B1*XP(1,1)-B2*XP(2,1)+UP
      XP(2,2)=SP(1,1)

```

```

AX=4.0
BX=63.75
CALL ROUND(XP(1,2),AX,BX)
YP=(A1-A0*B1)*XP(1,1)+(A2-A0*B2)*XP(2,1)
1  +A0*UP
CX=64.
DX=255./64.
CALL ROUND(YP,CX,DX)
DO 1 I=1,2
1  XP(I,1)=XP(I,2)
C(1)=C(2)
YP=YP/FX*3.0
RETURN
END

SUBROUTINE INPUT

C
C  RANDOM INPUT
COMMON/COM2/RM(1001),ITER,TES
TES=200.
ITER=301
NRANB=6
CALL RANBIT(NRANB)
CALL RCON1(35187269)
RMAX=(2.**NRANB-1.)/2.
DO19 I=1,ITER
RM(K)=IRAN(5)
19 RM(I) = (RM(I)-RMAX)/RMAX
RETURN
END

SUBROUTINE ROUND (A,AN,BN)
X=ABS(A)
S=A/X
IX=X*AN
XQ=IX
XQ=XA/AN
IF(XQ-BN) 1,2,2
1  A=S*XQ
RETURN
2  A=S*BN
RETURN
END

```

Real Time Programming

A digital filter implemented on a general-purpose computer, whether large or small, is said to be realized by real-time programming. The machine language version (translated from some higher level language) of the difference equations must execute quickly enough to meet the sampling rates imposed by the system specifications. In some applications the general-purpose computer will handle other calculations as well and will be "time-shared" to perform both duties. Other times a small process control computer can be dedicated solely to the digital filter calculations. An example system is shown in Fig. 2.1.

Generally speaking, future trends will be to design special-purpose computers to shoulder the digital signal processing tasks, and relieve the general-purpose computer for more complicated tasks which exploit its entire computational power as embodied by its versatile instruction set.

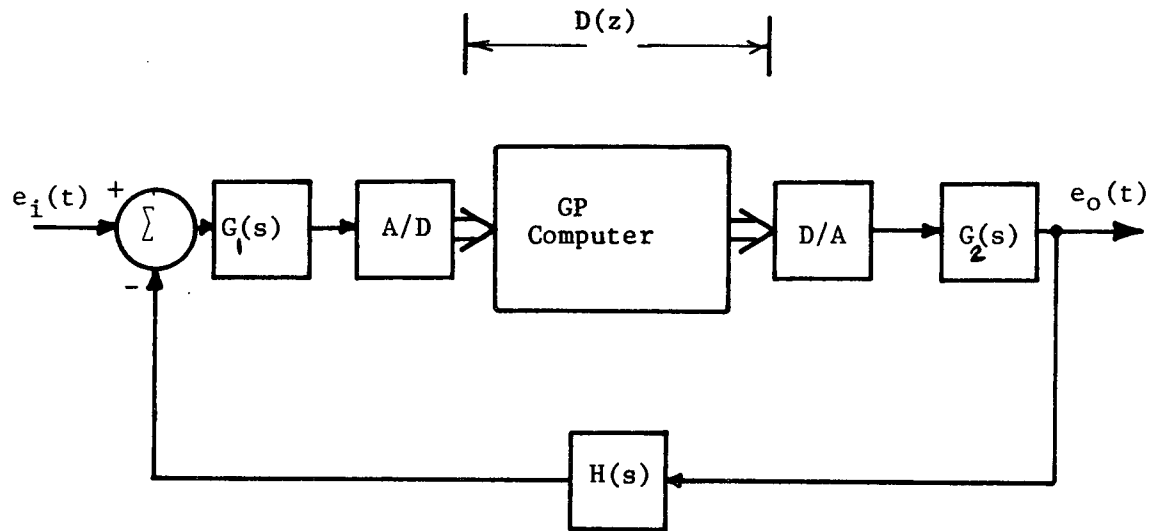


Fig. 2.1. A GP computer being used as a digital filter in a discrete control loop.

III. MINICOMPUTER IMPLEMENTATIONS

A minicomputer implementation of a digital filter as described in [22] will be discussed. Only one reference is used as a background since it is the only one that has been seen in the literature of digital filtering. It will be sufficient since any other minicomputer implementation would follow the guidelines presented.

Hardware Requirements.

The hardware used for the minicomputer implementation is shown in Fig. 3.1. It consists of a Honeywell H316 minicomputer with two 4096-word memory modules, a 10-bit analog-to-digital (A/D) converter, a 12-bit digital-to-analog (D/A) converter, a crystal-controlled real-time clock and the ASR-33 teletypewriter.

H316 minicomputer. The H316 is a GP minicomputer with a 16-bit wordlength. Arithmetic is performed in two registers, A and B, and it is a one-address machine with the A register serving as the accumulator which will be described in detail later. The memory is divided into sectors or pages of 512 words each, with the computer having the capability to reference any of the 512 words within a certain sector. Single-level indexing and/or multiple-level indirect addressing can be used to address words outside the current sector or the base sector.

With respect to the arithmetic instructions of the computers instruction set, there are two modes of operation: single precision and double precision. Each mode of operation may be entered by the use

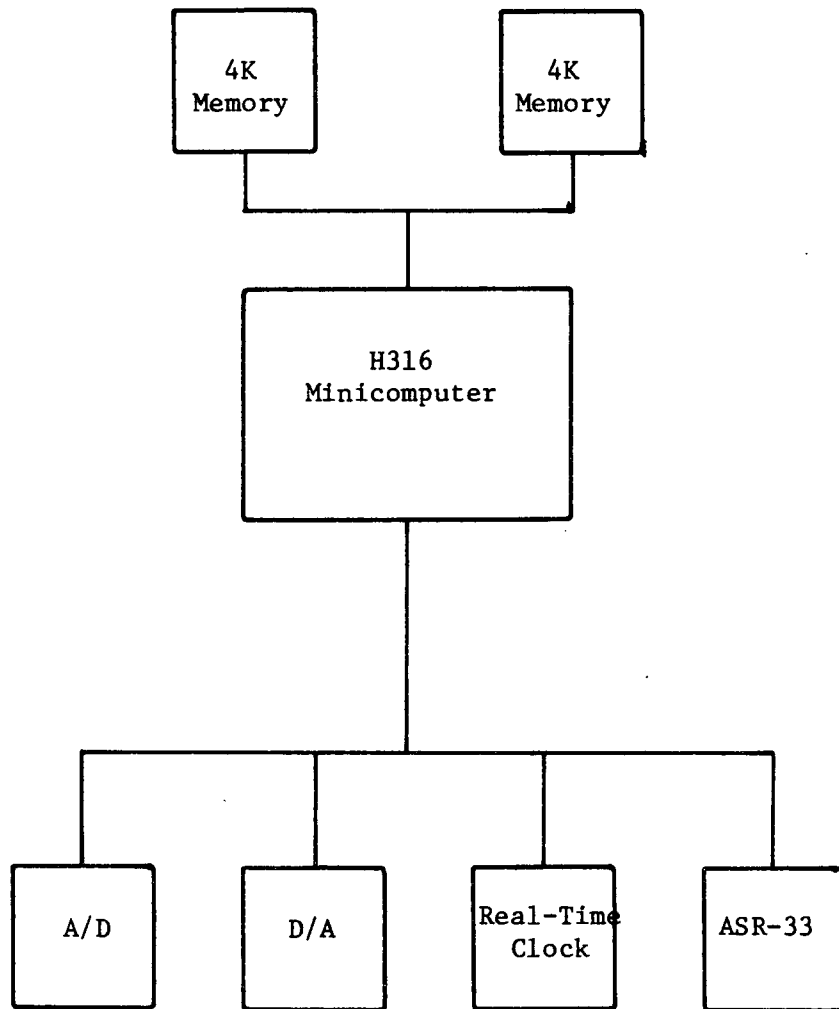


Fig. 3.1. Hardware used in minicomputer implementation.

of one instruction, "SGL" for single-precision arithmetic operations and "DBL" for double precision arithmetic operations. When operating in the single-precision mode, the A register is used solely as the accumulator. It is 16-bits long with the left-most bit being the sign bit and the 15-bits to the right being the most-significant through the least significant of the magnitude bits which are in a two's complement code. When operating in the double precision mode, the A and B registers are used as the accumulator with the sign bit being in the left-most bit position of the A register. The rest of the A register contains the 15 most significant bits of the double precision word with the 15 least significant bits being contained in the 15 right most bit positions of the 16 bit B register. The left most bit position of the B register does not take part in arithmetic operations.

When performing the "add" instruction which will have to be done many times in difference equations calculations, the contents of the addressed memory word are added to the contents of A leaving the sum in A for single-precision addition. If done in double-precision the contents of the addressed memory word (two memory locations for double precision) are added to the contents of the A and B registers and the sum left in them.

The same procedure occurs for multiply for the single or double precision mode. The addressed word in memory is multiplied by the word stored in the A or A and B registers and the product left in the A or A and B registers.

Fixed point arithmetic is used for all difference equation calculations. Since an imaginary binary point is assumed, after a multiplication instruction is executed, the computer shifts the product as required to align the binary point.

Input to the minicomputer is accomplished through 16 input bus lines into the A register. Several peripheral devices may be connected to this bus as inputs to the computer. In the case of the minicomputer implementation of a digital filter this bus inputs information from the A/D, ASR-33, and the real-time clock.

Output is accomplished through 16 output bus lines which are tied directly to the A register and always reflect its contents. For the digital filter implementation, the output device is the D/A or the ASR-33.

The different input devices are checked by the computer by placing a code unique to each device on the address bus.

Most peripheral devices are slower than the computer, thereby making the computer spend much of its time waiting for a peripheral device to perform its function. It is for this reason that it is practical to let the computer process other information while a particular peripheral is performing its I/O function. Then when the peripheral is finished, it can inform the computer and the computer can give it another command.

The method of informing the computer of the completion of a task is called an interrupt. When a peripheral interrupts the H316, it

finishes executing the instruction presently being performed and then performs a subroutine jump indirectly through a dedicated memory location. In short, the dedicated memory location contains the address of a subroutine to which the computer jumps when an interrupt occurs. Within this subroutine the computer may poll the peripherals to find out which one interrupted.

An A/D converter is used as the input interface element to the computer. The A/D which was interfaced to the H316 is a bipolar converter having a range of -10v to +10v. It has a 10-bit plus sign-bit output which is input into the most significant 10-bits of the A register.

The D/A converter accepts and transforms the binary output of the computer into an analog voltage. A hold register is employed so that the output voltage will remain constant until the next output occurs. The D/A used in the minicomputer implementation was built from Honeywell μ -Pac DTL logic. The converter is built from three cascaded Honeywell CE-071 four-bit converters which consist of a resistive ladder plus switching network.

To provide for a sampling rate other than that determined by the computers execution time, a real-time clock was employed. It initiates each cycle consisting of input, calculation, and output and is built as a peripheral which furnishes periodic (sample rate) interrupts to the computer. When an interrupt occurs, the computer goes through one cycle and then waits for the next interrupt before it goes through the

cycle again. This allows the operator of the minicomputer to obtain any desired sample rate.

Operating System

It is the purpose of the minicomputer implementation to be able to realize in real-time one of eleven different digital filter programming forms. It is the function of the operating system to set up, control, and possibly run diagnostic tests if something goes wrong, on the minicomputer and its peripherals.

A functional block diagram of the operating system (OS) is shown in Fig. 3.2. Solid arrows indicate a passing of control from one routine to another, while dotted arrows indicate a passing of parameters. Only one filter form is shown, but it should be remembered that eleven such forms are present with similar links to the operating system.

Briefly, to realize a digital filter, a particular form is picked and the parameters which determine the transfer function are input. The OS will then type back these parameters if desired. Once the filter form is set up, the OS is instructed to begin execution of that form.

Let us now discuss the different parts of the OS.

Executive. The executive routine (EXEC) initially types a question mark on the teletype. Whenever the question mark appears the operator types in one of four commands: MODIFY, LIST, RUN, or TEST. The first three refer to a particular programming form and are followed by a number between one and eleven. The TEST command refers to one of seven diagnostic routines and should be followed by a number from one to seven.

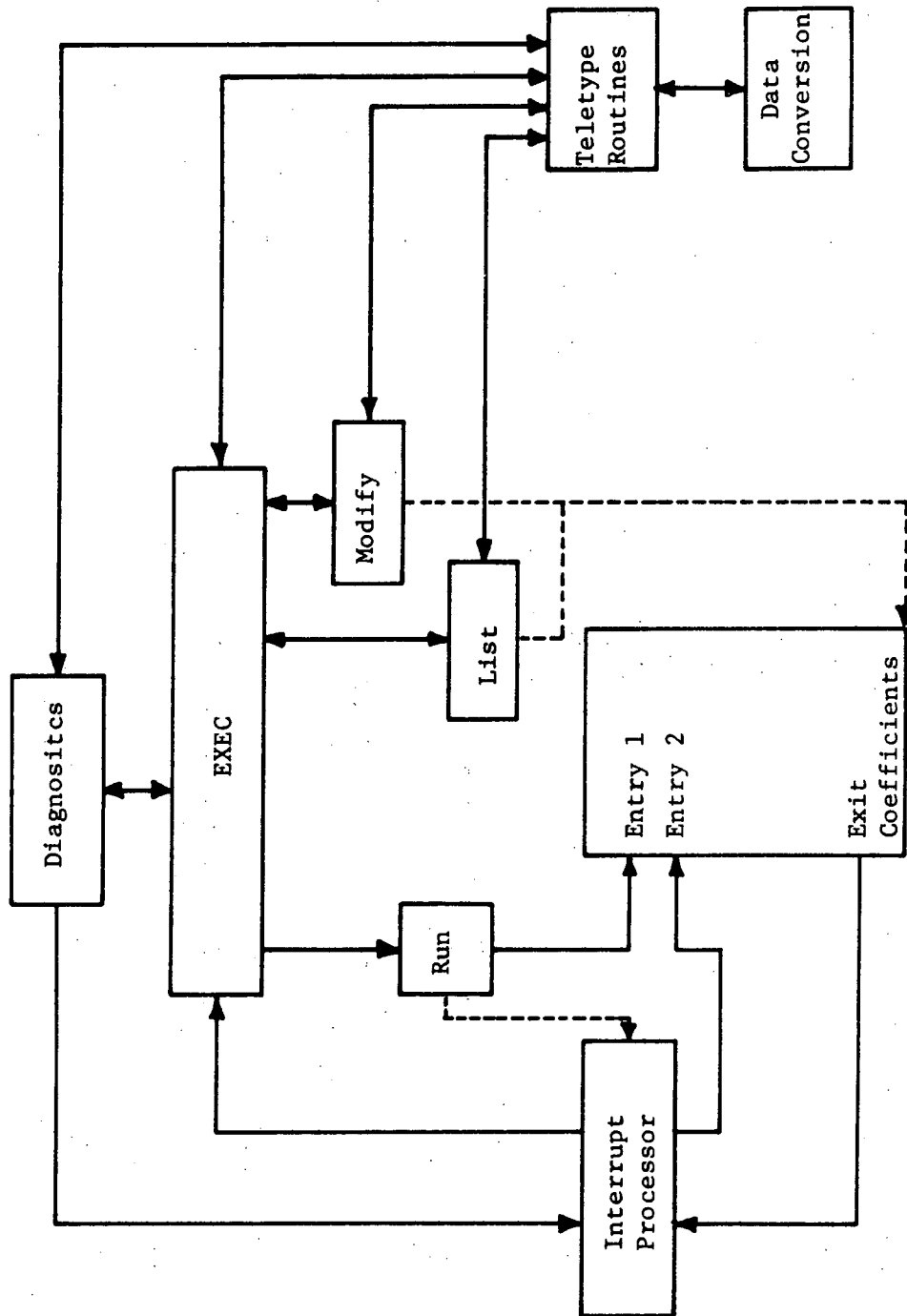


Fig. 3.2. Block diagram of operating system.

After one of the four commands is typed in EXEC turns control over to one of the four routines having the same name. Let us briefly discuss these routines.

1. Modify. The modify routine inputs the coefficients, quantization formats, and sample rate for a particular filter form. EXEC determines which of the eleven forms has been typed in following the command MODIFY, then transfers control to the modify routine, passing the filter form number as a parameter.

2. List. The list routine types out the coefficients of a programming form followed by the quantization formats and finally the sample period.

3. Run. To begin the filter processing the operator would type in RUN followed by the number of the form he desires to use.

RUN has a list of all entry points of the filter forms. When the RUN routine is entered it immediately obtains the address of the normal entry point and passes it to the interrupt processor which will need it at a later time. Then RUN selects the sample period which the user has specified for that filter form and outputs it to the RTC. Next, RUN sets the mask of the real-time clock (RTC) and teletype, starts the RTC, types out a question mark, and transfers control to the initialization entry of the filter form specified. The filter form makes its first pass and hangs up in the idle routines at the end. While in the routine the RTC should interrupt.

Interrupt processor. When the interrupt occurs, control is passed to the interrupt processor. This routine must identify what caused the interrupt and act accordingly.

User's interface. The user interface consists of the teletype routines plus the data conversion routines. The teletype routines are relied upon by all the other routines which have to communicate with the user. The teletype routines handling numerical data rely on the conversion routines to convert from decimal to binary and binary to decimal.

Diagnostics. Seven diagnostic routines are implemented in the OS to test the hardware and the software structure. One of these routines may be executed by typing in the request TEST followed by a number from one to seven. The errors that are checked for are divided into three categories: hardware errors, errors in the programming of the OS, and last, user errors.

This completes the discussion of the OS. We will now look at the assembly programs.

Assembly Programs.

Each of the eleven filter programming forms is realized by a separate subroutine which has the following format:

```
ENTRY
INPUT
CALCULATION
OUTPUT
TIME DELAY
PRECALCUATION
IDLE
EXIT
```


There are two entry points to each program: one being an initialization entry point which zero's the internal variables the first time through the program and the second a regular entry point that is entered everytime except the first. After entering the normal entry point a "start A/D" command is given and, while waiting on the input to become available, a partial sum is formed. As soon as the input arrives it is shifted to a correct format, multiplied by A_0 , and the sum is then completed. The sum is then quantized for output, presented to the D/A, then quantized in a different format for storage and feedback. If overflow is detected during quantization, the word is saturated, i.e. filled with the largest possible number.

After the output is complete, the internal variable must be shifted to perform time delay. Then the partial sum for the next pass is begun. Just enough of the formation of the partial sum is left for the next pass to occupy the arithmetic unit while waiting on the A/D. During the "idle" period, the RTC interrupts and the interrupt subroutine directs control back to the normal entry point.

The coefficients as well as the three shift instructions used in the quantizing routines are declared as external names so that they may be altered by the OS.

As an example of one of the eleven assembly language programs the assembly language program for a second order $D(z)$ in modified canonical programming form is shown below.

```

SUBR  MCAN1,ENT1
SUBR  MCAN2,ENT2
ENT   SHFT61,S1
ENT   SHFT62,S2
ENT   SHFT63,S3
ENT   COEF6,A0
BEL
* INITIALIZE INTERNAL VARIABLES
ENT1  CRA
      STA  XM1
      STA  XM2
*CALCULATE OUTPUT DIFFERENCE EQ.
ENT2  OCP  '41          START A/D
      LDA  XM1
      MPY  AL1
      DBL
      DST  TEMP
      LDA  XM2
      MPY  AL2
      DAD  TEMP
      DST  TEMP
      INA  '1041
      JMP  *-1          INPUT FROM A/D
                          WAIT FOR INPUT
S1    LRS  4
      STA  EI
      MPY  A0
      DAD  TEMP
      SGL
      STA  SGN
S2    LLS  9
      SSC
      JMP  OK1
      LDA  SGN
      CSA
      LSA  ='77777
      SRC
      TCA
OK1   OTA  '40
      JMP  *-1
*CALCULATE FEEDBACK DIFFERENCE EQ.
      LDA  EI
      MPY  ONE
      DBL
      DST  TEMP
      LDA  XM1
      TCA
      MPY  B1

```

```

      DAD    TEMP
      DST    TEMP
      LDA    XM2
      TCA
      MPY    B2
      DAD    TEMP
      SGL
      STA    SGN
S3    LLS    3
      SSC
      JMP    OK2
      LDA    SGN
      CSA
      LDA    ='77777
      SRC
      TCA
* PERFORM TIME DELAY
OK2   STA    XM
      LDA    XM1
      STA    SM2
      LDA    XM
      STA    XM1
      ENB
      NOP
      JMP    *-1
XM1   DBP    0

EO    BSS    1
XM    BSS    1
EI    BSS    2
XM2   BSS    2
TEMP  BSS    2
SGN   BSS    1
AO    OCT    10000
AL1   OCT    -22753
AL2   OCT    7357
B1    OCT    3146
B2    OCT    231
ONE   OCT    10000
      END

```

Experimental Results

Experimental results were obtained of the minicomputer implementation previously described.

First it realized the transfer function of a Euler integrator

$$D(z) = \frac{1}{1 - z^{-1}} \quad (\text{II-1})$$

in the direct form at its maximum sampling rate (5.5 KHz). The response was obtained for an input square wave and as wished, the output was a triangular waveform with a fine-grained stair-stepped appearance.

Secondly it realized the transfer function of a digital differentiator

$$D(z) = 1 - z^{-1} \quad (\text{II-2})$$

in the direct programming form. It's response to a triangular waveform, a square wave, was as expected.

Lastly, the transfer function of a digital oscillator was realized

$$D(z) = \frac{1}{1 - 2\cos(2\pi fT)z^{-1} + z^{-2}} \quad (\text{II-3})$$

where T is the reciprocal of the sample rate (5.5 KHz) and f is the frequency of oscillation. The equation was programmed with $b_1 = -1.75$ which resulted in an output frequency of 450 Hz as predicted by Eq. (II-3).

Experimentation demonstrated that the direct and canonical forms had the highest maximum sampling rate. The direct, canonical and modified canonical forms have minimum sample intervals of less than 200 μ -secs. The modified direct form has a minimum interval of about 225 μ -secs.

The parallel and cascade forms have a minimum interval of about 275 μ -secs, while the remainder of the forms have intervals of approximately 300 μ -secs.

The number of instructions required to implement the filter forms (including coefficient storage), ranges from 98 for the canonical to 109 for the modified cascade. The entire operating system occupies approximately 5000 memory locations including indirect links in the base sector.

This concludes the discussion on minicomputer implementations of digital filters. From this discussion it was seen that a minicomputer can be adapted well for a real-time digital filter implementation; in fact, much better than the larger SP computers because of the smaller size. We will now look at even a smaller digital computer implementation, that of implementation by special-purpose computers.

IV. SPECIAL-PURPOSE COMPUTERS

It is obvious to one that for most realizations of a digital filter, the general purpose computer and the minicomputer approach have several disadvantages. The most easily seen disadvantage, as previously mentioned, is the wasted hardware incurred because of the relative simplicity of the difference equations that must be calculated for a realization. It is for this reason that the special-purpose computer approach to realization is taken for a majority of the applications of digital filtering. It will be shown in the following discussion of special-purpose (SP) computer realizations that they are the most economical (hardware wise) and demonstrate a great amount of versatility.

The realization techniques by SP computers will begin with the very earliest method, which was sample and hold devices with analog networks and conclude with present day commercial models that are available on the market.

Implementation by Sample and Hold Devices with Analog Networks.

The first SP computer realizations of a digital filter were by sample and hold devices with analog networks [23]. There are two main ways of realizing a digital filter in this manner with them being: 1) series discrete data networks, and 2) feedback discrete data networks. There is a third way of realization which is a combination of the above two

that will not be discussed since it is felt it is not essential to illustrate the realization technique.

The series discrete data network which is used for the realization of the discrete transfer function of a digital filter is shown in Figure 4.1. The transfer functions of the two systems in Fig. 4.1 are related by

$$D(z) = G_{ho} G_c(z) \quad (4-1)$$

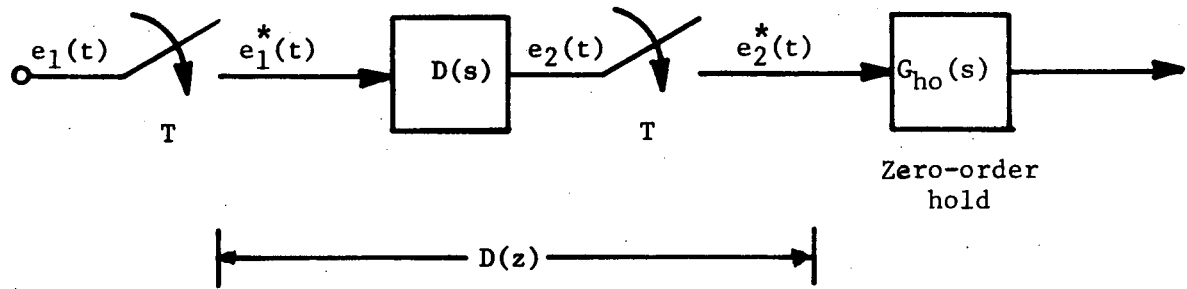
Since the transfer function of the zero-order hold is $(1 - e^{-Ts})/s$, it can be obtained from Eq. (4-1) that

$$Z\left[\frac{G_c(s)}{s}\right] = \frac{1}{1 - z^{-1}} D(z) \quad (4-2)$$

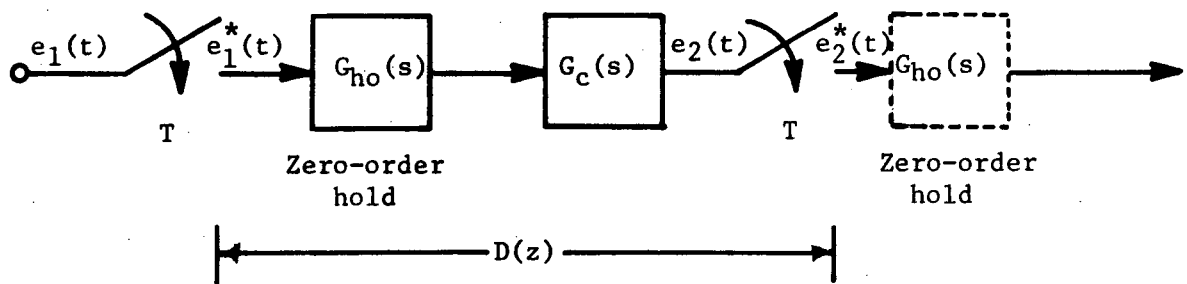
From this it is seen that given a specific $D(z)$, the transfer function $G_c(s)$ of the discrete-data network can be determined from Eq. (III-2) by taking the inverse z -transform.

If $G_c(s)$ is to be an RC realizable transfer function, all the poles of $G_c(s)$ must be simple and lie on the negative real axis of the s -plane with the exception of the origin and infinity. The zeroes of $G_c(s)$ may be located anywhere in the s -plane. Therefore, $G_c(s)/s$ can be expanded into the following form by partial fraction expansion:

$$\frac{G_c(s)}{s} = \frac{A_o}{s} + \sum_{k=1}^m \frac{A_k}{s + s_k} \quad (4-3)$$



(a) Digital filter.



(b) Equivalent series discrete-data network.

Fig. 4.1

where A_0 and A_k are constants and $s_k = [k = 1, 2, 3, \dots, m]$ are simple negative real poles. The z -transform of Eq. (III-3) is

$$z \left[\frac{G_c(s)}{s} \right] = \frac{A_0}{1 - z^{-1}} + \sum_{k=1}^m \frac{A_k}{1 - e^{-s_k T} z^{-1}} \quad (4-4)$$

which has simple positive real poles inside the unit circle $|z| = 1$, with only one pole at $z = 1$. Comparing Eq. (4-4) with Eq. (4-2), it is seen that in order for $G_c(s)$ to represent an RC network, the discrete transfer function $D(z)$ must have the following properties:

- (1) The number of poles of $D(z)$ must be equal to or greater than the number of zeroes of $D(z)$.
- (2) The zeroes of $D(z)$ are arbitrary in location.
- (3) The poles of $D(z)$ must be simple, real and positive, and lie inside the unit circle $|z| = 1$ in the z -plane.

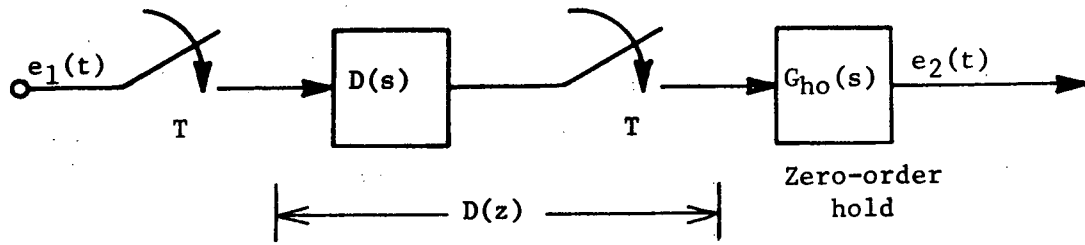
It can be shown that for a feedback discrete-data network, the feedback structure with a zero-order hold and the RC network shown in Fig. 4.26 is equivalent to the digital filter of Fig. 4.2a [23].

The transfer function of the two systems are related by

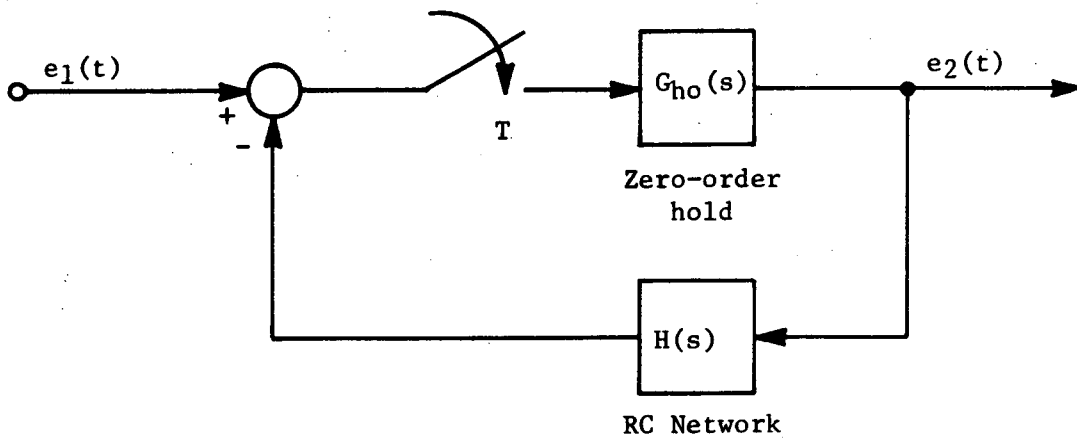
$$D(z) = \frac{1}{1 + G_{ho} H(z)}$$

or

$$z \left[\frac{H(s)}{s} \right] = \frac{1}{1 - z^{-1}} \left[\frac{1 - D(z)}{D(z)} \right] \quad (4-5)$$



(a) Digital filter.



(b) Equivalent feedback discrete-data network.

Fig. 4.2

In order to realize $H(s)$ by an RC network, the discrete transfer function $D(z)$ must have the following properties:


- (1) $D(z)$ must have the same number of poles and zeroes.
- (2) The poles of $D(z)$ are arbitrary.
- (3) The zeroes of $D(z)$ must be simple, real, positive, and lie inside the unit circle of the z -plane.

The reasons behind these restrictions are discussed in detail in [23] and therefore are not repeated here.

In summary it may be said that the listed restrictions and limited flexibility of the above implementation techniques make their use and practicality almost negligible.

Hybrid Implementation.

After observing the difference equations of several of the programming forms which may be used for the realization of a digital filter, such as the direct and canonical forms, one can see that these equations might be computed by a device which performs the arithmetic functions of addition, subtraction, multiplication and time delay. This suggests the use of summing amplifiers, constant multipliers and delay elements for the implementation. Also, knowing that digital filters may be implemented by SP and GP computers suggests the realization of a digital filter by hybrid techniques, with hybrid meaning that analog and digital methods are used for the implementation [10]. For most hybrid realizations digital techniques are used for analog-to-digital (A/D)



conversion, digital-to-analog (D/A) conversion and time delay with analog techniques used for the arithmetic functions of addition, subtraction, and multiplication. This type realization exploits the best and most natural functions of both analog and digital elements to eliminate a majority of the restrictions placed on the realization of the previous section of the literature which used sample and hold devices with analog networks. Another advantage of the hybrid realization which will be illustrated shortly is that once a $D(z)$ is obtained, the filter can be realized directly from it. This will eliminate the need for additional mathematical manipulation required for the derivation of an s-plane transfer function from the $D(z)$ as was required by the previous implementation technique.

The hybrid implementation of a digital filter fits itself to a majority of the programming forms of a given $D(z)$. For simplicity sake, we will look at the hybrid realization of a $D(z)$ in two programming forms; the direct form and the canonical form. These two forms usually have the simplest difference equations which must be implemented for their realization.

For most hybrid designs an A/D converter functions as the sampling device as well as an interfacing element for the input to the filter. Integrated circuit buffer registers are usually used to store previous values (in digital form) of an intermediate variable that is internal to the controller. D/A converters provide the interface for the output of

the filter. Variable resistors at the input of a summing amplifier are usually used to adjust the coefficients of the compensation function continuously over a wide range of values. The equivalent of a zero-order hold device is realized at the output of the filter as a result of the digital data-storage elements within the hybrid unit.

The hybrid filter is designed, in this case, to realize almost any compensation function up to three zero's over three poles in the z domain with any sampling frequency up to several thousand hertz. If a filter of order higher than three is required, several second or third order filters are cascaded until the desired order is obtained. This is usually done over constructing a higher order filter because of the greater coefficient sensitivity for a higher order filter. It is seen that the hybrid controller is extremely versatile and can be used to realize a wide range of sampled-data compensation functions; thus, as previously mentioned, it is not necessary to redesign the unit to change the compensation function.

To obtain an insight into the design of a hybrid digital filter for a given $D(z)$, a hybrid implementation will be derived for the direct and canonical programming forms.

Let us look at the direct form realization first. A second order transfer function for a realizable digital filter can be written as

$$D(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} = \frac{E_0(z)}{E_1(z)} \quad , \quad (4-6)$$

where the coefficients a_1 and b_1 are real numbers and $z = e^{sT}$.

Equation (4-6) can be rewritten as

$$E_0(z) = a_0 E_1(z) + a_1 z^{-1} E_1(z) + a_2 z^{-2} E_1(z) - b_1 z^{-1} E_0(z) - b_2 z^{-2} E_0(z) \quad (4-7)$$

or in the time domain as the difference equation

$$e_0(kT) = a_0 e_1(kT) + a_1 e_1(kT - T) + a_2 e_1(kT - 2T) - b_1 e_0(kT - T) - b_2 e_0(kT - 2T) \quad (4-8)$$

where $f_s = 1/T$ is the sampling frequency. Fig. 4.3 is a block diagram of the hybrid realization of the second order direct programming form with double lines denoting digital information and the single lines analog information. A detailed explanation of the components used for the realization will be given after the block diagram of the canonical form is given.

In order to realize the canonical form, we have previously seen that for a second order $D(z)$, the following difference equations must be realized.

$$m(kT) = e_1(kT) - b_1 m(kT - T) - b_2 m(kT - 2T) \quad (4-9)$$

$$e_0(kT) = a_0 m(kT) + a_1 m(kT - T) + a_2 m(kT - 2T) \quad (4-10)$$

The block diagram that results for a canonical hybrid realization is shown in Fig. 4.4. It is easily seen how this form results after careful consideration of the form of the two difference equations that must be implemented.

The actual construction of the canonical implementation will now be discussed in more detail. This form was chosen to be discussed because of several advantages it has over the direct programming form. One of the primary reasons, which is obvious from Figs. 4.3 and 4.4, is that the canonical form is much more economical. For example, the direct form requires two A/D converters whereas the canonical requires one. Also, if n is the order of the numerator of the $D(z)$ being realized and m the denominator, the direct form requires $n + m$ delays whereas the canonical requires the greater of n and m . This is also the case for D/A converters. It can be said that in general anytime a $D(z)$ is to be realized by a hybrid realization, choose the programming form which requires the least hardware.

Let us now discuss the analog and digital components of a hybrid implementation. Observing the canonical realization of Fig. 4.4, the first step in a hybrid realization is the conversion of analog information into digital information by the use of an A/D. There are two types of A/D's which might be used, the first being the successive approximation type and the second the count-up-to type.

Fig. 4.5a is a block diagram of a successive approximation type A/D converter. The comparator compares the A/D input signal m_0 (the

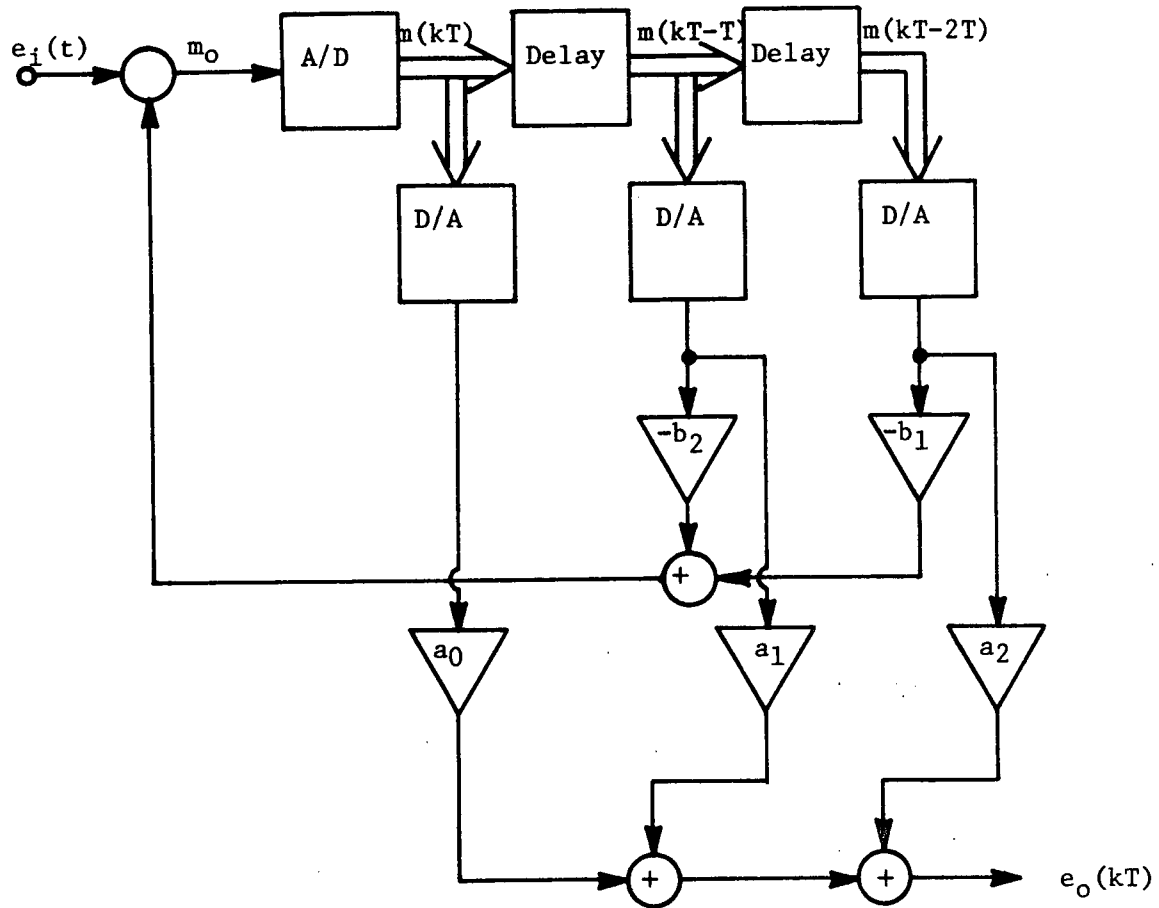
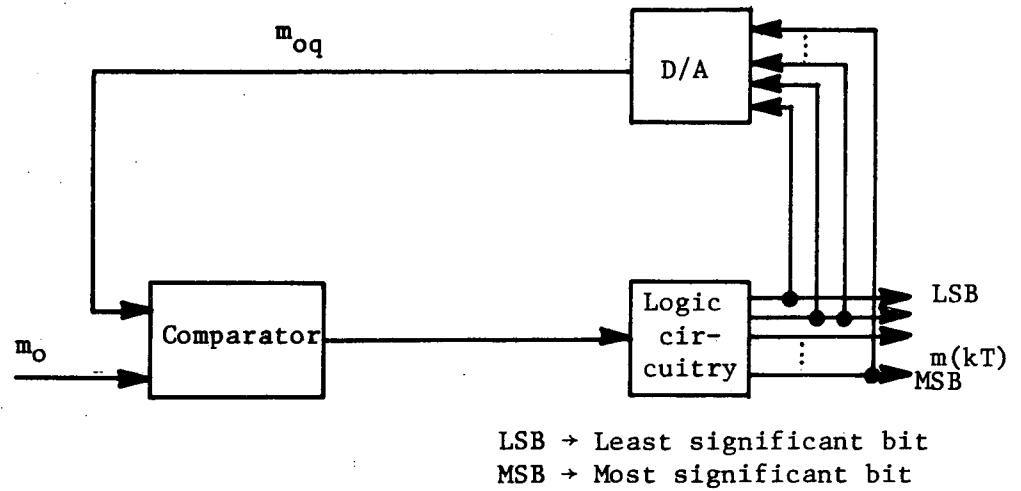
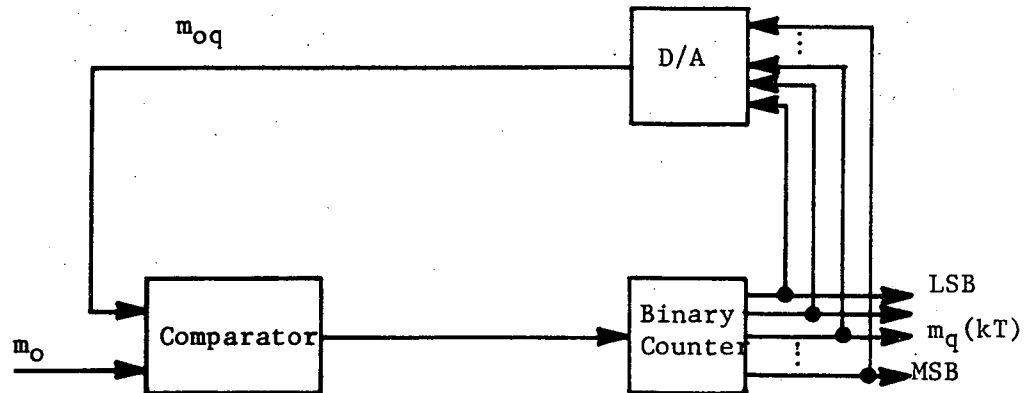


Fig. 4.4. Block diagram of a hybrid realization of a digital controller in canonical programming form.



(a) Successive approximation type A/D converter.



(b) Counter type A/D converter.

Fig. 4.5. A/D converters.

signal m_0 is the output of the feedback summing amplifier as shown in Fig. 4.4) to the quantized output m_{0q} of a D/A converter. The signal m_{0q} is determined wholly by the digital word $m(kT)$ which is in sign magnitude code. The logic circuitry is programmed to "search for" or "home-in on" the analog signal m_0 . Successive approximation type converters are faster (commercial models are available that will convert an analog signal to an 8 bit sign magnitude code approximation in 200 nsec) than the counter types which will be described shortly, therefore they are used in a majority of applications.

Fig. 4.5b is a block diagram of the counter type A/D converter. In general it is slower than the successive approximation type converter and is therefore used when the input analog signal m_0 is of lower frequencies. This type of converter operates on the principle of letting a binary counter count until its output decoded through a D/A converter is equal to the input signal. When this occurs the counter ceases operation and its output bit sequence ($m(kT)$) at this time is a sign magnitude code approximation of the analog input m_0 . Before the converter can be ready for the next conversion the binary counter will have to be set such that all its bits are logic "0", with this being completed before each conversion.

Of the two types of converters discussed above, it is recommended that the successive approximation type be used for digital filter implementations because of its ability to handle high frequency input signals.

The time delay indicated by Eqs. (4-9) and (4-10) can be provided by clocked flip-flops. This means the information at the inputs of the flip-flop is held or stored until the clock terminal is pulsed, at which time the information that is on the input is transferred to the output terminals.

The D/A converters shown in Fig. 4.4 are the ladder type networks commonly used in constructing D/A's. From the Figure it is seen that the contents of the time delays constitute previous values of $m(kT)$. These digital words are decoded by the D/A converters into analog signals and are then available for further analog processing; that is, multiplication and summation.

From Eqs. (4-9) and (4-10) it is seen that the a_i and b_i are real numbers and may be positive or negative. This suggests the use of operational amplifier circuits to perform the arithmetic operations of multiplication and summation. From Fig. 4.6, which illustrates in detail the summation and multiplication techniques for a second order implementation in canonical programming form, it is seen that the algebraic sign of the coefficients is obtained by inverting the output m_{iq} of the i^{th} D/A (multiplying by -1) so that m_{iq} and $-m_{iq}$ are available. From Fig. III-6 it is also seen that the magnitude of the coefficients is obtained by variable input resistors to an operation amplifier.

Any hybrid digital filter implementation should generally be implemented in the same procedure as the above. If there are variations, they are usually small, and are left up to the individual designer.

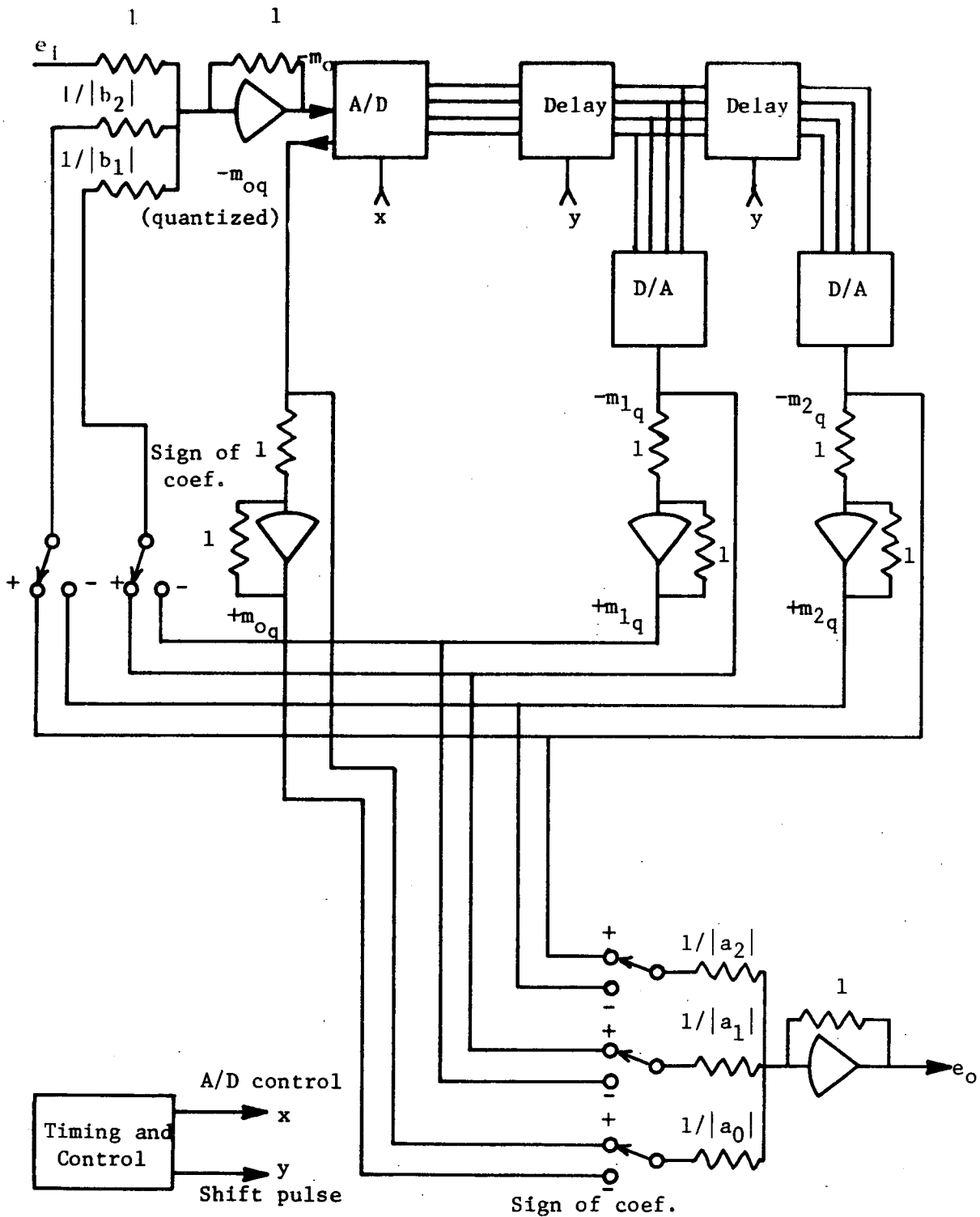


Fig. 4.6. Hybrid implementation of a second-order digital compensator.

Digital Implementation.

The adaptation of a small SP computer for the implementation of digital filters only seems natural after a careful consideration of the requirements that must be met for the realization of the difference operations of a particular programming form. Let us consider the requirements of the difference equations of an arbitrary programming form since the requirements would hold for all forms. Let our choice be the difference equations required for the realization of a $D(z)$ in the modified canonical programming form. These equations for a second order $D(z)$ are shown below:

$$e_o(kT) = a_0 e_i(kT) + \alpha_1 m(kT - T) + \alpha_2 m(kT - 2T) \quad (4-11)$$

$$m(kT) = e_i(kT) - b_1 m(kT - T) - b_2 m(kT - 2T) \quad (4-12)$$

Observing the above equations we see that for them to be physically realized a device must be used which can add, subtract, multiply, perform time delay, truncate and provide data storage. A device which can do all of this is a small SP computer. All of the components of a digital computer can be organized into four main functioned units as shown in Fig. 4.7.

Considering the functional requirements of a digital filter, we see that the Arithmetic Unit of the computer can perform the addition, subtraction, multiplication, and the truncation required for the realization of the digital filter. The Memory can be used to perform

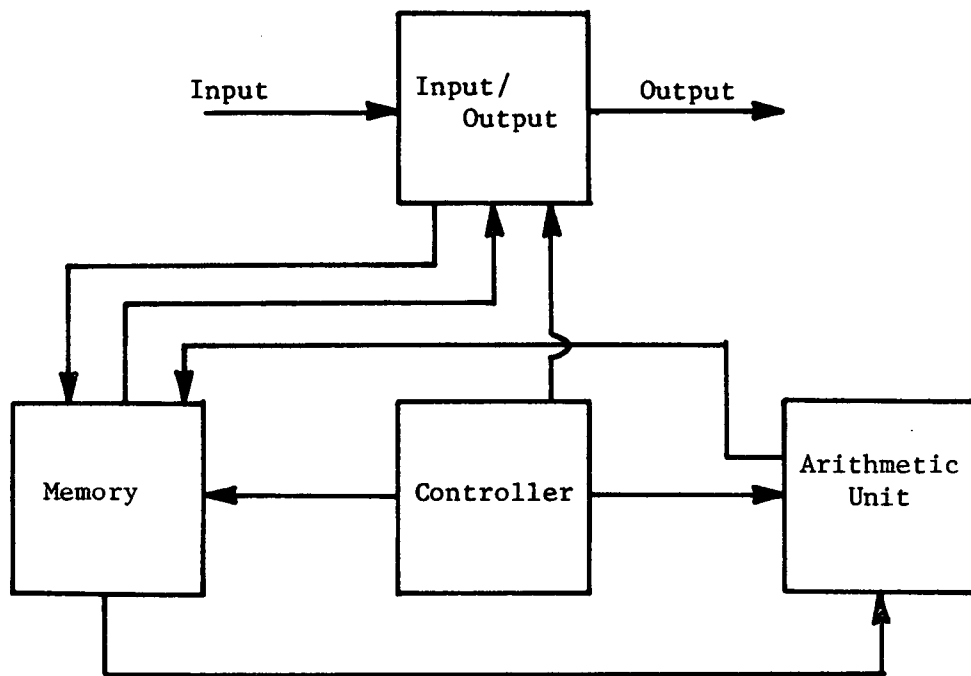


Fig. 4.7. Four functional units of a digital computer.

the time delay, coefficient storage, internal variable storage, and input/output storage required for a filter implementation. The Input/Output functional block of the digital computer will accommodate the A/D and D/A converters if required for interface for a digital filter realization. The Control Unit functional block of the digital computer will accommodate the controller for the digital filter and the data transfer logic which insures correct routing of data for the realization of the required difference equations. At this point we see that all of the arithmetic, storage, input/output, and control requirements necessary for the implementation of a digital filter have all been incorporated into the four functional blocks of a digital computer; i.e., a digital filter may be realized by a small SP computer and its functional diagram is shown in Fig. 4.8 [7]. The computer is said to be small because it will be designed to only calculate the difference equations for a particular programming form. Another reason the resulting SP computer is small is because of reduced word lengths that are required thereby requiring less hardware.

Now that it has been shown that a small SP digital computer can be used for the realization of a digital filter, it is now appropriate to discuss how one would go about designing a SP computer realization of a digital filter and the considerations that must be made while doing this.

The design consists of three parts: first, the determination of quantization levels in the computer (input quantizing, round-off errors,

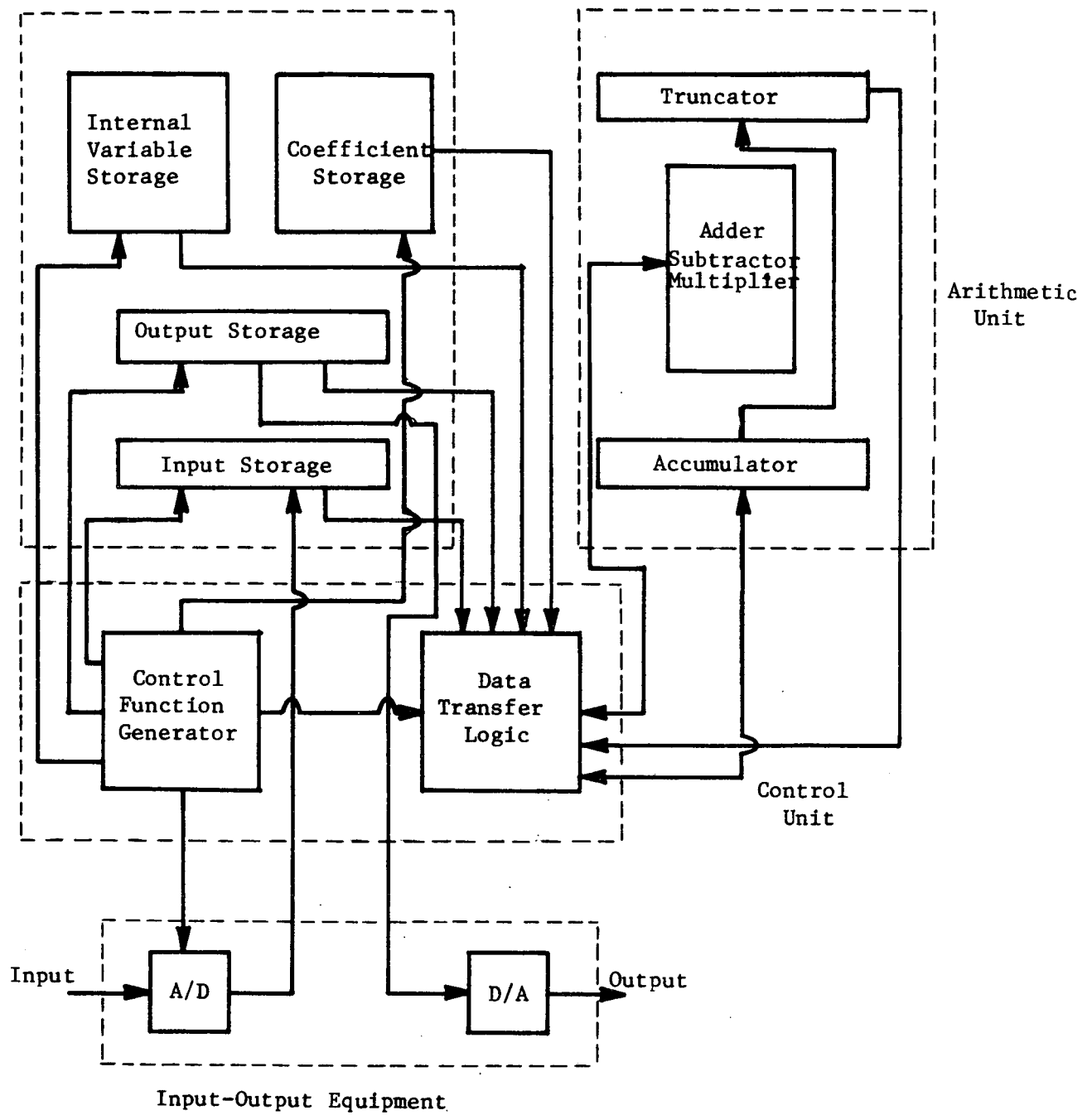


Fig. 4.8. Functional diagram of a digital filter.

and filter coefficient quantizing); second, the logical design of the computers components; and third, the interconnection of the computers components to implement the above mentioned quantization levels.

The first step in the design procedure, the determination of quantization levels, will not be covered here since it would be a reiteration of earlier sections of this work. As a reminder though, one could use several techniques to do this, among them being the CAD program and different quantization error analysis techniques.

Next consider the second step, which has not been presented previously and will be discussed in depth.

The second step in the design of any SP computer implementation of a filter, as previously mentioned, consists of the logical design of the computer components. All of the components will be grouped into four main component groups with these being, 1) Input/Output components, 2) Arithmetic components, 3) Memory components and 4) Controller components. The discussion will begin with the design of the input/output components.

1. Input/Output Components

When designing the input/output equipment for a digital filter the first decision to be made is that of what type information will be the input and output of the filter, i.e. is the input/output of the filter going to be in analog or digital form.

If the input and output of the filter is in digital form the design problem will usually be minimal since the filter normally operates on

digital inputs and outputs in digital form. The only problem that might be encountered if the application of the filter is such that it will have digital input/output is that of synchronization between the device that is supplying the filter its input and the filter. Provisions must be made when designing the filter such that it will accept a digital input word from a device which is supplying it.

For many applications of digital filtering, the filter will be operating in an environment composed mainly of analog signals which will necessitate input/output interface elements for the filter. These interface elements will be A/D converters for the input to the filter and D/A converters for the output of the filter as shown in Fig. 4.9.

First let us discuss the selection of an A/D converter. The first decision to be made is how fast should its conversion speed be. In general this is dictated by the frequency of the input signal or the maximum sample rate of the filter. If the input analog signal is of high frequencies and the filter is to sample at a high rate the successive approximation type converter previously discussed usually would be better since it is faster than the counter type. Next, a consideration of the number of bits required for the digital approximation of the analog signal would be required. This can be determined by knowing the maximum analog input voltage (V_{\max}) to the filter and the maximum allowable quantization step length h . If these two parameters are known the number of quantization steps required can be determined by

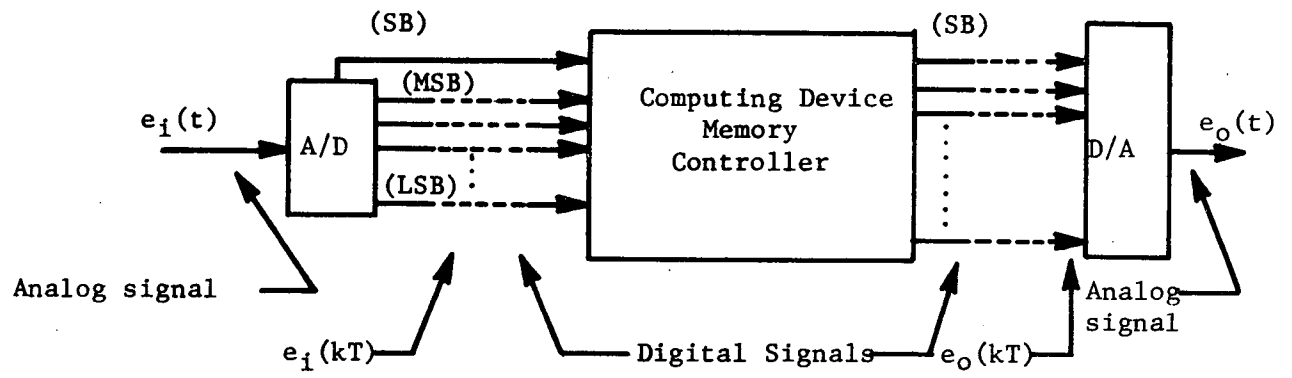


Fig. 4.9. A digital filter with its interface elements.

$$\text{Number of Quantization Steps} = \frac{V_{\max}}{h} = QS \quad (4-13)$$

Knowing QS, the number of magnitude bits (n) required for the A/D converter can be chosen if n is the smallest integer such that

$$(2^n - 1) \geq QS \quad (4-14)$$

Now that the bit length and speed of the A/D converter is known, the next step is the construction or selection of a commercial A/D. In selecting a commercial A/D, there are many distributors available. All one has to do to find them is to thumb through an electronics or computer oriented magazine. A few pointers to remember when selecting commercial A/D converters are that the faster their conversion speed, the more they cost, and if internal reference voltages are supplied they are also more costly than when the user supplies the references.

If a user were to construct his own A/D converter, its cost would also be determined by its bit length, speed, and the construction of circuits to supply reference voltages if they are not already available. There is abundant material available on how to construct successive approximation and counter type converters.

Choosing the output interface element, the D/A converter, is generally one of the easier design tasks of designing a digital filter. If a D/A converter is required, to construct an analog approximation of the digital output of a filter, there are two basic types from

which to choose. One type has a current output proportional to the magnitude of the digital word it is converting and the other type has a voltage output. Which type is chosen depends on the application of the digital filter being designed. Most D/A converters are constructed from resistive ladder type networks.

There are many commercial types of D/A's on the market today and their manufacturers are easily found by simply, as for A/D's, thumbing through computer oriented magazines.

There are several characteristics of D/A's that must be considered when buying one. One consideration is if the D/A will have to be unipolar or bipolar. Some D/A's will only operate in the unipolar or bipolar mode, and others can be connected to operate in both. Another important characteristic to consider is the speed in which the conversion is made and the settling time of the D/A. If operation of the digital filter is in the low frequency range, not as much attention will have to be paid to this as if it were operating in a high frequency range. As anyone might speculate, the faster the conversion time and settling rate, the higher the price.

2. Arithmetic Unit

When designing the arithmetic unit of a digital filter, the first major decision that has to be made is that of parallel arithmetic operations or serial arithmetic operations. Both schemes have been proposed and both have their advantages and disadvantages. The question as to which method is better can only be determined by the designer and his use for his filter.

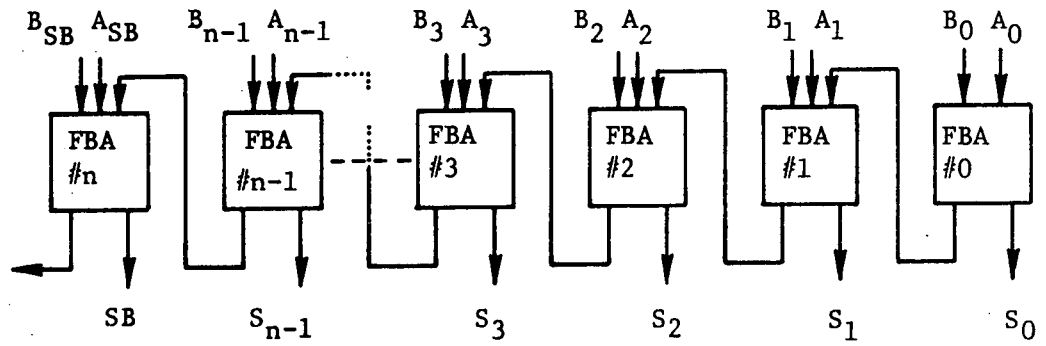
In general parallel arithmetic operations are used when there is a desire for speed and serial arithmetic operations are desirable for a minimum of hardware. Techniques of performing parallel and serial arithmetic operations will be considered in more detail shortly.

The second decision that must be made when designing an arithmetic unit of a digital filter is that of what type binary code to use. Since the arithmetic unit must be able to add, subtract, multiply and perform truncation, it would seem logical to use a signed one's complement or a signed two's complement binary code. The reason for using these codes over a straight signed magnitude code is because, with them, subtraction can be performed by an addition process. Also, multiplication can be performed by a shifting and addition process using these codes. Of the signed one's complement code and the signed two's complement code it seems that a majority of the time the two's complement code is used.

Now an example of an arithmetic unit organized in a parallel and serial fashion will be presented, starting with a parallel organization.

This particular arithmetic unit is able to add, subtract, multiply, and perform truncation. The code used by the computer is a signed two's complement code.

The arithmetic unit performs the addition and subtraction process of two n bit words, A and B , by use of n full-binary-adders (FBA) connected in the configuration shown in Fig. 4.10. If addition is to be performed ($A + B$), A and B are applied to the input of the adder circuit and the output will be the sum of A and B . If the arithmetic operation



$$A = A_{SB}, A_{n-1} \cdots A_0$$

$$B = B_{SB}, B_{n-1} \cdots B_0$$

Fig. 4.10. Full-binary-adders used for addition and subtraction of two's complement binary numbers.

(A - B) is performed, then A is applied as it is to the input of the adder and B is two's complemented and applied to the input. The resulting output of the adder is (A - B).

Now that it has been demonstrated that we can add and subtract two words in two's complement code with FBA circuits, an accumulator will be defined and it will be shown that with an accumulator and shift registers, all the arithmetic operations of addition, subtraction and multiplication can be performed.

A binary accumulator consists of a register, which stores a binary number (the augend) in signed-magnitude form and upon receiving another binary number (the addend) in the same form adds the second number to the first and then stores the sum in the register. The logic diagram of a parallel binary accumulator is shown in Fig. 4.11. Each flip-flop in the accumulator functions as a modulo 2 counter. The augend is initially stored in the accumulator and, during the addition process, each flip-flop counts parallel incoming pulses representing the addend bits and generates a carry pulse to the next significant bit when the flip-flop changes its state from 1 to 0.

To illustrate the use of the parallel binary accumulator in performing the multiplication and summation process, observe the difference equation

$$e_o(kT) = a_0 e_i(kT) + c_1 m(kT - T). \quad (4-15)$$

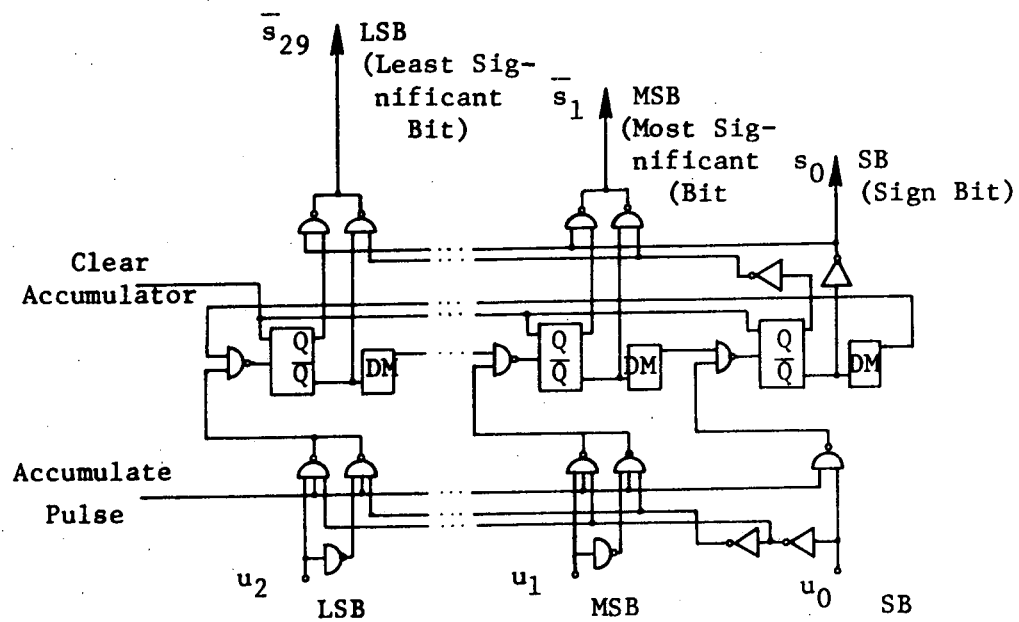


Fig. 4.11. Parallel binary accumulator.

Let

$$\begin{array}{ll}
 a_0 = +1.25 = +(1.01)_2 & \text{coefficient} \\
 c_1 = +0.50 = +(0.10)_2 & \begin{array}{l} + \rightarrow 0 \\ - \rightarrow 1 \end{array} \\
 e_i(kT) = +6. = +(110.)_2 & \text{variable} \\
 m(kT - T) = -3. = -(011.)_2 &
 \end{array}$$

then

$$\begin{array}{rcl}
 a_0 e_i(kT) = & \begin{array}{r} +110. \\ \times +1.01 \\ \hline +110. \\ + 00.0 \\ + 1.10 \\ \hline + 111.10 \end{array} & c_1 m(kT - T) = \begin{array}{r} -011. \\ \times +0.10 \\ \hline -000. \\ - 01.1 \\ - 0.00 \\ \hline -001.10 \end{array}
 \end{array}$$

$$\text{and } e_o(kT) = + 111.10 - 001.10 = +110.0 = +(6.)_{10} .$$

A simplified block diagram of a shift register and parallel binary accumulator implementation of the example difference equation is included in Fig. 4.12. The following sequence of events is offered to explain the operation of this implementation.

1. Set the accumulator output to zero.
2. Load $c_1[+0.10]$ and $m(kT - T)[-011.]$ into the two shift registers.
3. Apply an accumulate pulse, a shift pulse, another accumulate pulse, a second shift pulse, and a third accumulate pulse (abbreviate this sequence by asasa; the output of the accumulator is now $c_1 m(kT - T)[-001.10]$).
4. Clear the shift registers.

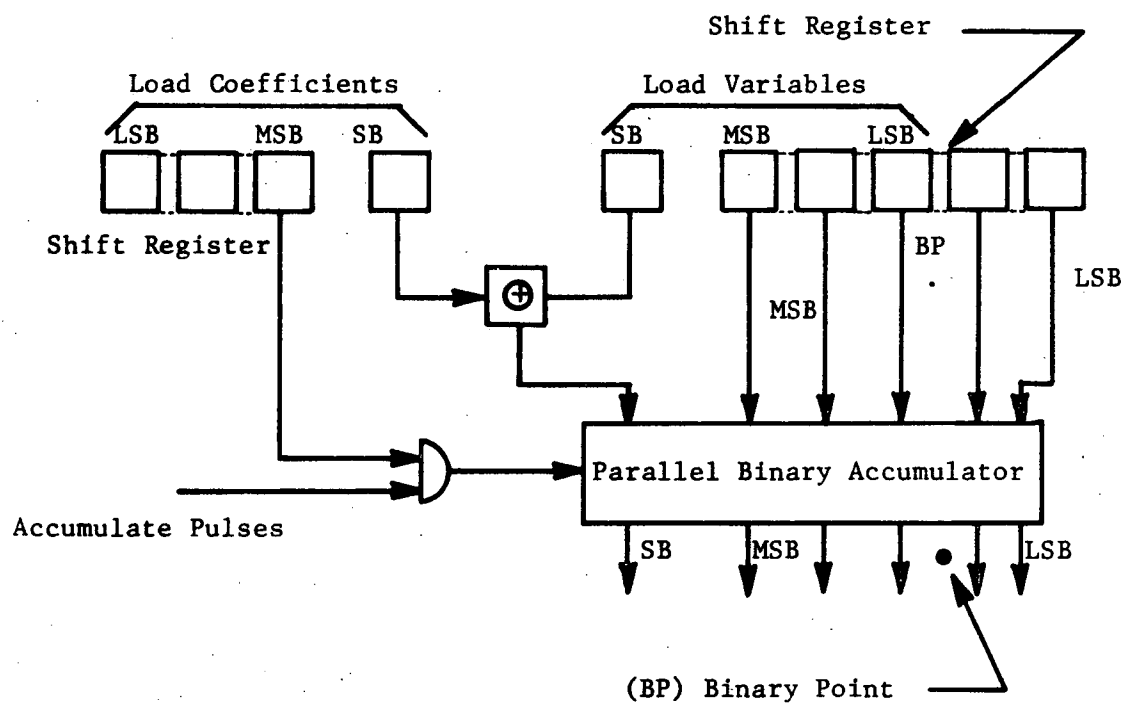


Fig. 4.12. Implementation of an example difference equation.

5. Load $a_0[+1.01]$ and $e_1(kT)[+110.]$ into the two shift registers while leaving $c_1m(kT - T)$ shared at the accumulator output.
6. asasa; the accumulator output is now $a_0e_1(kT) + c_1m(kT - T)[+110.00]$.

Thus the difference equation is implemented.

There are other designs for accumulator of Figs. 4.11 and 4.12. One design which is used frequently uses FBA's and clocked J-K flip-flops. This accumulator design is shown in Fig. 4.13. This accumulator operates in the same manner as the previously discussed design when used in calculating difference equations. The FBA's are used to add the new input to the accumulated sum already in the accumulator (stored at the output of the JK flip-flops) and once the new sum is obtained an accumulate pulse is applied which clocks the J-K flip-flops and causes the input bits of the flip-flops to be transferred to the outputs where it will be stored until the next accumulate pulse arrives.

As stated previously, serial arithmetic may also be used for digital filter implementation. Let us consider it now.

Serial arithmetic is used mainly for two reasons: 1) there is a savings in hardware which will be demonstrated shortly and 2) serial arithmetic provides for an increased modularity and flexibility in the digital circuit configurations.

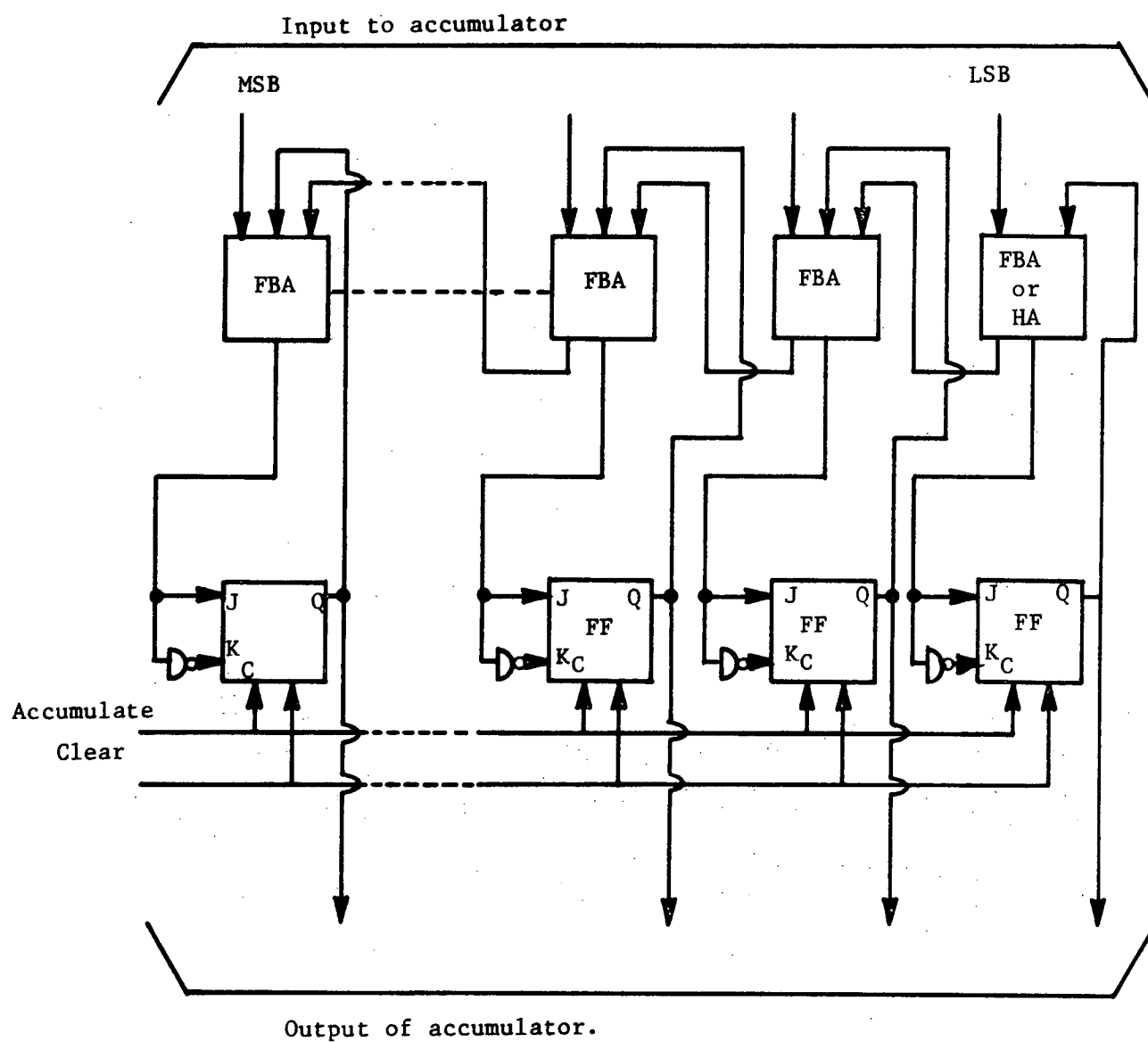


Fig. 4.13. Parallel binary accumulator constructed from FBA's and clocked J-K flip-flops.

The two's-complement representation of binary numbers is appropriate for digital filter implementation using serial arithmetic because additions may proceed, starting with the least significant bits, with no advance knowledge of the signs or relative magnitudes of the numbers being added.

The serial arithmetic unit, as the parallel implementation, must be able to add, subtract, and multiply and truncate.

The block diagram of a serial adder (subtractor) is shown in Fig. 4.14.

Briefly it operates in the following manner.

1. All registers and the delay flip-flop are cleared.
2. Words A and B, which are to be added or subtracted, are shifted into the shift registers to the left of the FBA with their binary points aligned. The shifting stops when the LSB of A or B reaches the LSB position in the register. All registers are now properly set to perform the addition (subtraction) process.
3. Now shift all registers to the right and the delay flip-flop for the carry at the same time. The proper sum of difference will be shifted into the register on the right.

If subtraction is being performed by the above circuit, the subtrahend will have to be two's complemented before it is loaded into its appropriate register. This operation can be performed with a simple

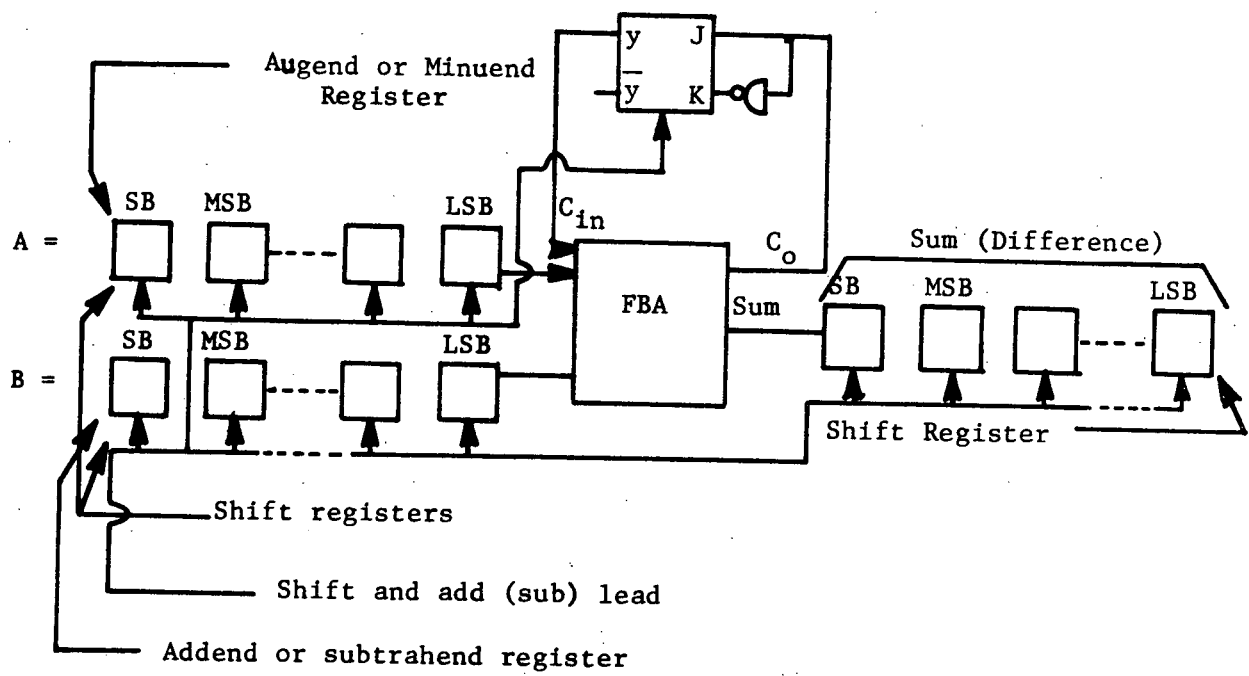


Fig. 4.14. A serial adder (subtractor) circuit.

sequential circuit which, for each input word, passes unchanged all initial least significant bits up to and including the first "1" and then inverts all succeeding bits. The circuit that will do this is depicted in Fig. 4.15 [18].

A serial multiplier configuration is shown in Fig. 4.16. This multiplier will only multiply two positive two's complement numbers, which does not restrict it, since the circuit of Fig. 4.15 can be used to two's complement any negative number that must be used in the multiplication process. Also, one's complement numbers may be multiplied by this circuit since if they are negative they may be complemented by a simple circuit to obtain the positive form. If this is done for a two's complement or a one's complement code, the sign bit of the resulting product will have to be retained and if it is negative, the resulting product term will have to be complemented appropriately.

Briefly, the serial multiplier of Fig. 4.16 works in the following manner. There are three shift registers Z, X, and Y, a serial adder and a half adder. The sign bit leads the serial word as indicated by the subscripts of the letters Z, X, and Y. All bits except bits X_1 and Y_1 of registers X and Y form a combined shift register. Register Z is also a circulating register.

Initially, the multiplier is stored in register Y with the sign bit in Y_1 . Next the multiplicand is serially transferred to register Z with the sign bit in Z_1 . Sign bits Z_1 and Y_1 do not move during the succeeding shifting of the contents of registers.

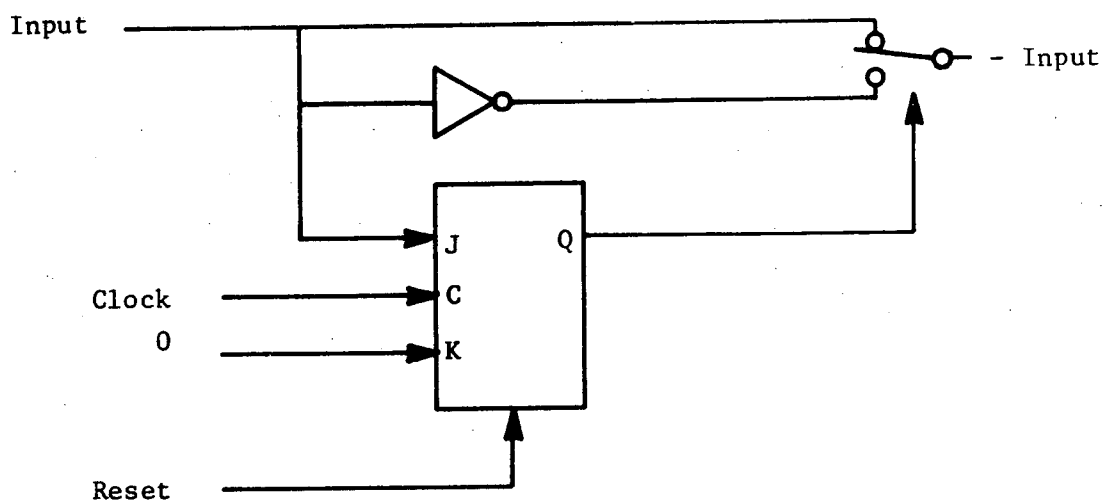


Fig. 4.15. Serial two's complementer.

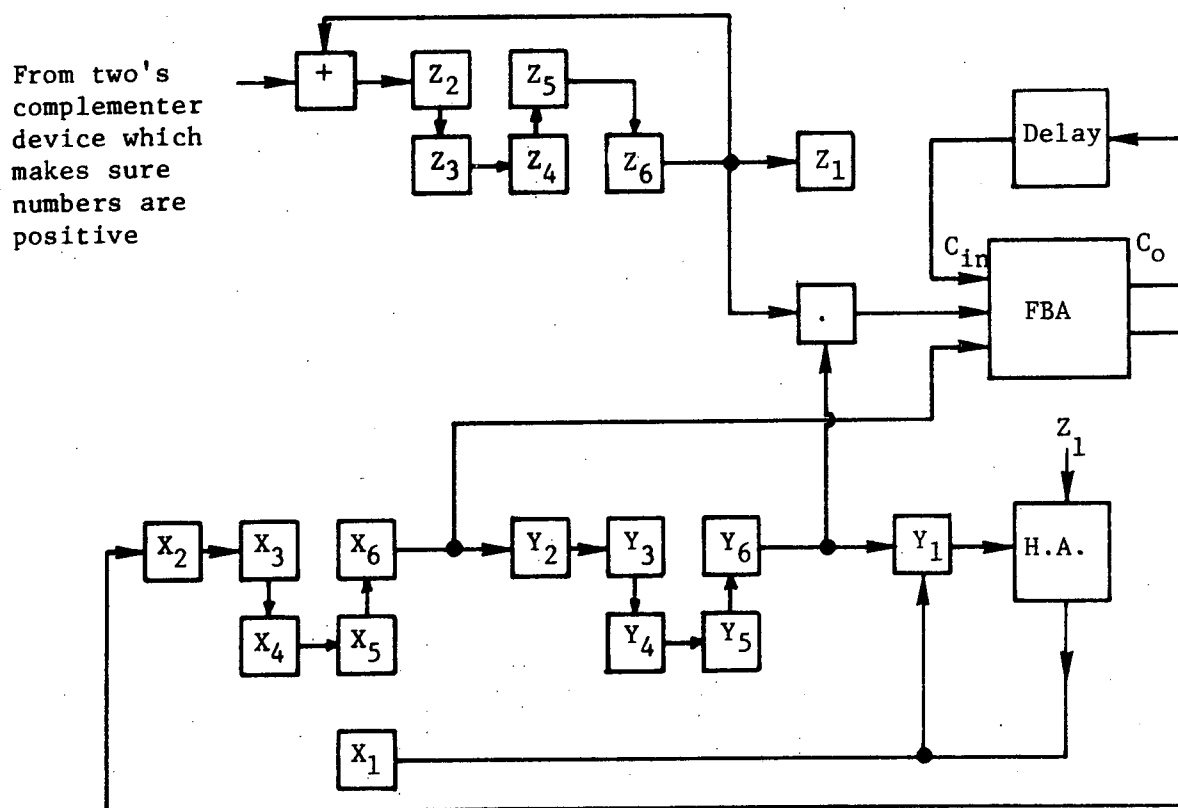


Fig. 4.16. A serial multiplier.

The value of 1 or 0 of the least significant bit of the multiplier in Y_6 determines whether or not the number bits of the multiplicand are to be added; and the addition, if there is one, is carried out. Also during this addition time, the circulating register Z restores its original contents and the partial product is serially inserted into register X, occupying bits X_2 to X_6 . The combined register is then shifted 1-bit to the right; during this shift, any carry bit left in the delay flip-flop is shifted into X_2 , and the least significant bit is now at Y_2 . The next addition begins at X_6 but not at bit Y_2 .

After the right shift, the least significant bit of the multiplier in Y_6 is lost, as Y_1 is not a part of the combined register. Y_6 now contains the second least significant bit of the multiplier, which possibly could initiate another multiplication.

This process of addition and right shifting continues until all multiplier number bits are shifted out of the combined register. After this, the product is available in the combined register with the most and the least significant halves of the product being stored respectively in the X and Y registers. When there is no round-off the sign bit will be in Y_1 . When there is round-off, 1 is inserted into bit X_6 , and the sign of the product is inserted into bit X_1 . After this the number in register X is in desired order [24].

When using the multiplier described above, the product term addition required for the completion of the difference equation calculation process can be handled by the serial adder previously described.

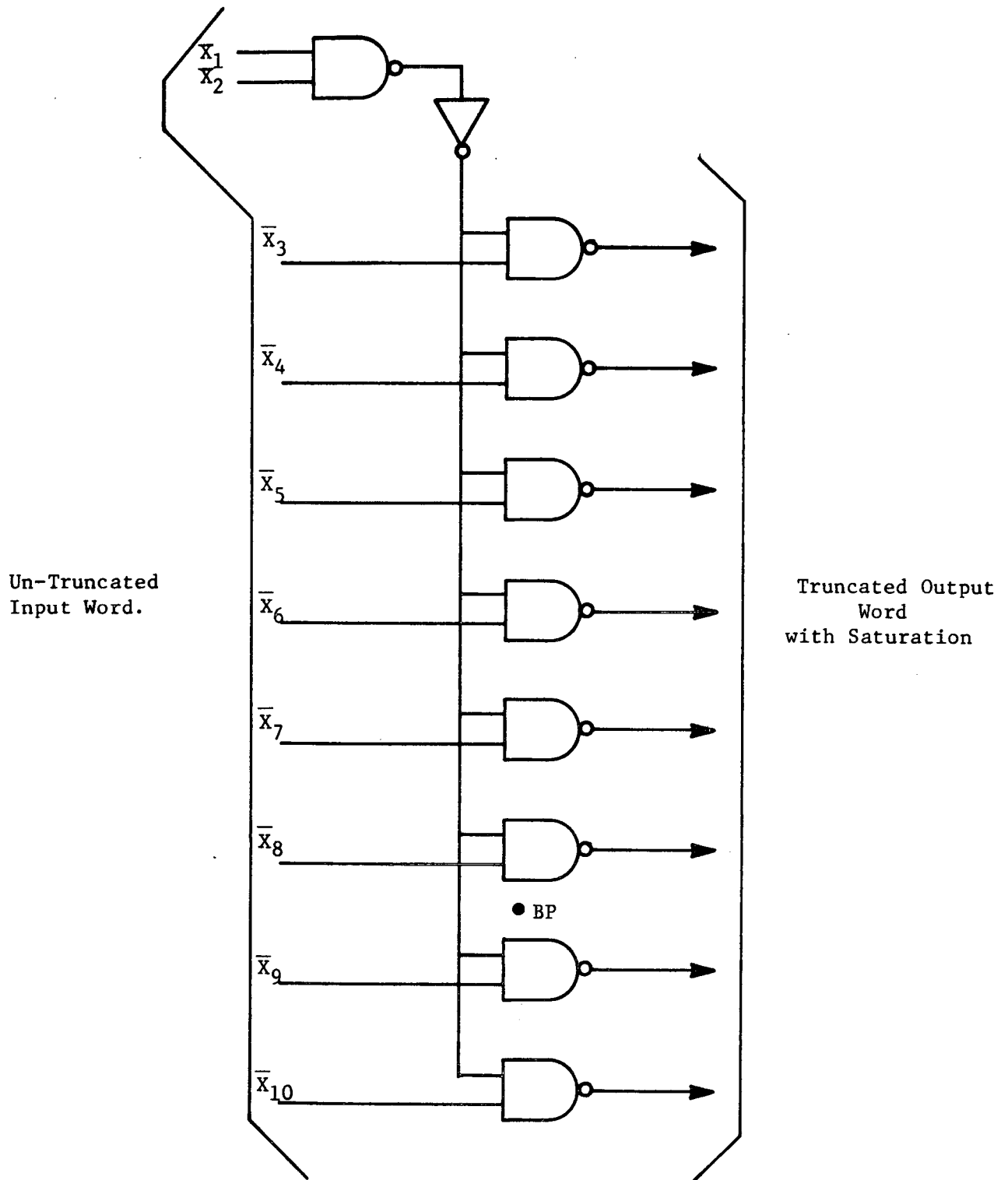
There are other techniques of parallel and serial multiplication that will not be discussed here. One technique which makes parallel multiplication much faster is the Wallace technique described in [25]. Likewise, there is a serial multiplication technique presented in [18] which makes serial multiplication much faster.

Usually when a difference equation is calculated by a digital filter, it is reduced in bit length before it is stored in memory or the output register. It is the purpose of the reduction logic of a digital filter to do this. Most reduction logic either performs truncation or round-off with most being truncation type.

The circuitry required for reduction logic is usually simple since it is usually a combinational logic circuit. Shown in Fig. 3.14 is reduction logic which will truncate a signed magnitude binary word with 14 magnitude bits (8 to the left of the binary point and 6 to the right) such that it has 6 bits to the left of the binary point and 2 to the right. The lower four bits that are truncated are omitted from the truncated word and anytime the untruncated word has a weighted bit in one of the two most significant bit positions, every bit of the truncated word will be a weighted representation, i.e. the truncated word is saturated.

3. Memory Design.

The memory of a digital filter is used for coefficient storage, interval variable storage which performs time delay, input word storage and output word storage.



BP → Binary Point

Fig. 4.17. Reduction Logic

There are two types of memory that are usually used for digital filter implementations: 1) flip-flop type memories, and 2) read-only-type memories (ROM).

A majority of the time flip-flop type memories are used for internal variable storage ($m(kT - T)$, $m(kT - 2T)$, ..., $m(kT - nT)$) which is required for the implementation of the time delays and also for the storage of the input and output variables of the filter. Shown in Fig. 4.18 is a flip-flop type memory for the storage of $m(kT - T)$ and $m(kT - 2T)$ required for the realization of a second order $D(z)$ in the modified canonical programming form. The word lengths in this figure are 8 magnitude bits with 6 of them to the left of the binary point. For this type memory construction a time delay is performed when the J-K flip-flops are clocked; $m(kT)$ becomes $m(kT - T)$ and simultaneously $m(kT - T)$ becomes $m(kT - 2T)$. The input and output storage registers are constructed the same way as the first column of flip-flops in Fig. 4.18.

ROM type memories are used mostly for coefficient storage. The coefficient values are loaded only once into the ROM and they are read out when they are needed for an arithmetic operation.

Digital filters may also employ simple single pole, double throw switches for coefficient storage. This is usually done for versatility as the coefficients may be changed by a simple toggle of a switch. The only disadvantage of this is that the coefficients cannot be changed

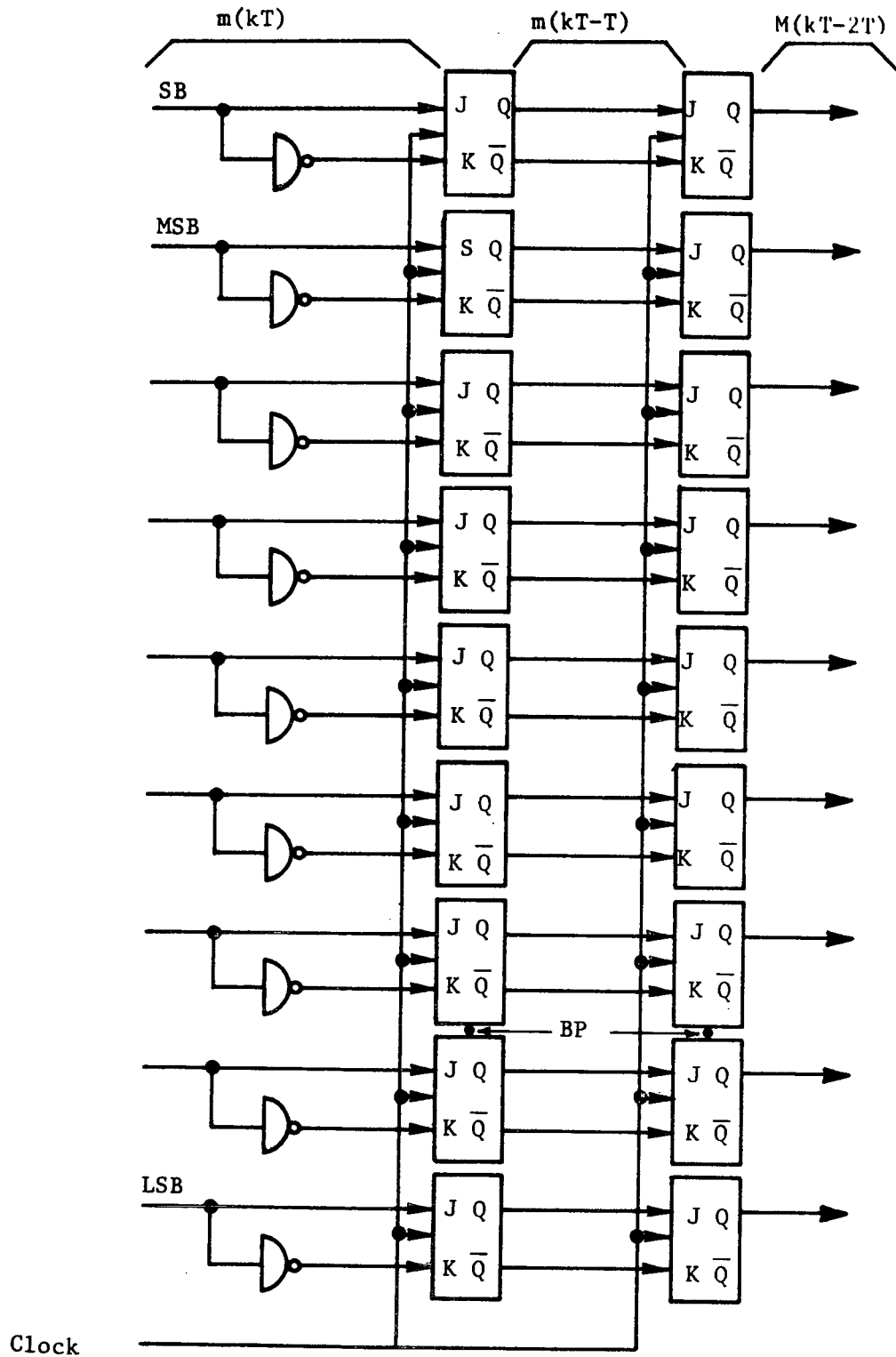


Fig. 4.18. Internal variable storage for a second order $D(z)$ in modified canonical programming form.

while the filter is operating in real-time. It must be stopped and then started over again. The reason for this is that the switches can't all be manually thrown during one sample period which would be required.

4. Controller Design.

The controller of a digital filter ensures the proper calculation of the difference equations being realized by the filter. As an example of what a controller must do, let us consider the calculation of the difference equations of a second-order $D(z)$ in the modified canonical programming form. The difference equations are shown below

$$e_o(kT) = a_0 e_i(kT) + c_1 m(kT - T) + c_2 m(kT - 2T) \quad (4-16)$$

$$m(kT) = e_i(kT) - b_1 m(kT - T) - b_2 m(kT - 2T) \quad (4-17)$$

Eq. (4-16) will be calculated first as it is desirable to obtain an output as soon as the input is available to the filter. One possible operation sequence that the controller may assume for a parallel arithmetic realization using an accumulator is listed below:

1. Activate the A/D so that $e_i(kT)$ can be obtained.
2. Shift the $m(kT - T)$ and $m(kT - 2T)$ storage registers to perform time delay.
3. Clear the accumulator and its associated shift registers.
4. Load c_1 and $m(kT - T)$ into their respective shift registers.

5. Perform the multiplication $c_1 m(kT - T)$.
6. Clear the accumulator shift registers.
7. Load c_2 and $m(kT - 2T)$ into the shift registers.
8. Perform the multiplication $c_2 m(kT - 2T)$.
9. Clear the shift registers.
10. Load a_0 and $e_i(kT)$ into the shift registers.
11. Multiply $a_0 e_i(kT)$. The output of the accumulator now contains

$$a_0 e_i(kT) + c_1 m(kT - T) + c_2 m(kT - 2T) = e_0(kT).$$
12. Clear the accumulator and the shift registers.
13. Load $-b_1$ and $m(kT - T)$ into the shift registers.
14. Multiply $-b_1 m(kT - T)$.
15. Clear the shift registers.
16. Load $-b_2$ and $m(kT - 2T)$ into the shift registers.
17. Multiply $-b_2 m(kT - 2T)$.
18. Clear the shift registers.
19. Load $e_i(kT)$ and 1.0 into the shift registers.
20. Multiply $1.0 e_i(kT)$. The output of the accumulator is now

$$e_i(kT) - b_1 m(kT - T) - b_2 m(kT - 2T) = m(kT).$$

$$\begin{matrix} \cdot \\ \cdot \\ \cdot \end{matrix}$$

Repeat same process again.

To ensure the correct sequence of operations that must be performed and the correct transfer of data such that there will be data available when required, the controller is divided into two parts: 1) the control

function generator and 2) the data transfer logic. The control function generator is simply a logic circuit which has as its output a sequence of pulses (with their timing and spacing very important) which controls the operation of the arithmetic unit, input/output, and memory. The data transfer logic is simply combinational logic circuitry which, under command from the control function generator, transfers data to and from the memory, input/output, and arithmetic unit.

The first step in designing any controller is the selection of an operation sequence, such as the previous example. After this is complete, the control function generator and then the data transfer logic may be designed.

Now that an approach has been presented by which all the functional components of a digital filter may be designed, the only remaining design consideration remaining is the interconnection of all four functional components such that they may function as a digital filter.

The interconnection of the functional units may be done in numerous ways. No specific approach will be given here since, in most cases, each designer of a digital filter has what he thinks is his own unique and novel way to interconnect the functional components. In general, the connections of Fig. 4.8 must be made in as modular fashion as possible for possible LSI realizations. If they vary it will be because of programming form variation and order of $D(z)$ variation.

Now that a basic SP computer implementation of a digital filter has been discussed, it will be in order to discuss variations of the SP computer implementation.

Implementation by Microprogrammable SP Computer.

The three previously discussed implementation techniques all have limitations, the most noticable of these being that each type implementation will realize a $D(z)$ in only one programming form. It would be advantageous to design a digital filter which would realize a $D(z)$ in any of the eleven previously discussed programming forms.

In answer to the question that may arise as to why is one programming form better than the other; it can be shown, as discussed previously, that for a particular $D(z)$ with set coefficients and sampling rate, different programming forms have different quantization errors. In general, for a particular $D(z)$ that must be realized, it is desirable to choose the programming form that introduces the least quantization error, therefore necessitating the need for a digital filter implementation that can realize a $D(z)$ in several programming forms. The implementation approach taken to do this is a microprogrammable design as discussed in [12].

It will be the purpose of this section to give a discussion of the computer organization and not to go into too much detail about the logic design since it is not necessary for an understanding of the operation of the microprogrammable implementation.

The prime function of the SP computer is the realization of a second-order digital filter in a choice of digital filter programming forms. As for the previous implementations this calls for the solution of the appropriate difference equations and entails the operations of addition, multiplication and time delay. The SP computer is binary, synchronous and parallel, with the calculations to be done using 2's complement arithmetic. It is a stored program computer, i.e., the Control Unit looks up the sequence of instructions in the memory, initiates arithmetic operations and causes operands or immediate results to be transferred between the Arithmetic Unit and the Memory, as required by the program instructions. Synchronization of the computer operations and generation of control pulses is the task of the Control Unit.

To increase the sampling rate of the filter, a high speed Wallace Multiplier is employed as described in [12]. Also a rapid-access memory will be used for the program memory to aid in decreasing the worst case delay between the time of input sampling and its corresponding output.

A block diagram illustrating the four basic functional units of a digital computer is as shown in Fig. 4.7. Therefore, just as for the previously described digital implementation of a digital filter, the organization of the SP computer will be divided into these four units. A detailed block diagram of the SP computer is shown in Fig. 4.19. The functional blocks in Fig. 4.19 which make up the Input/Output Unit

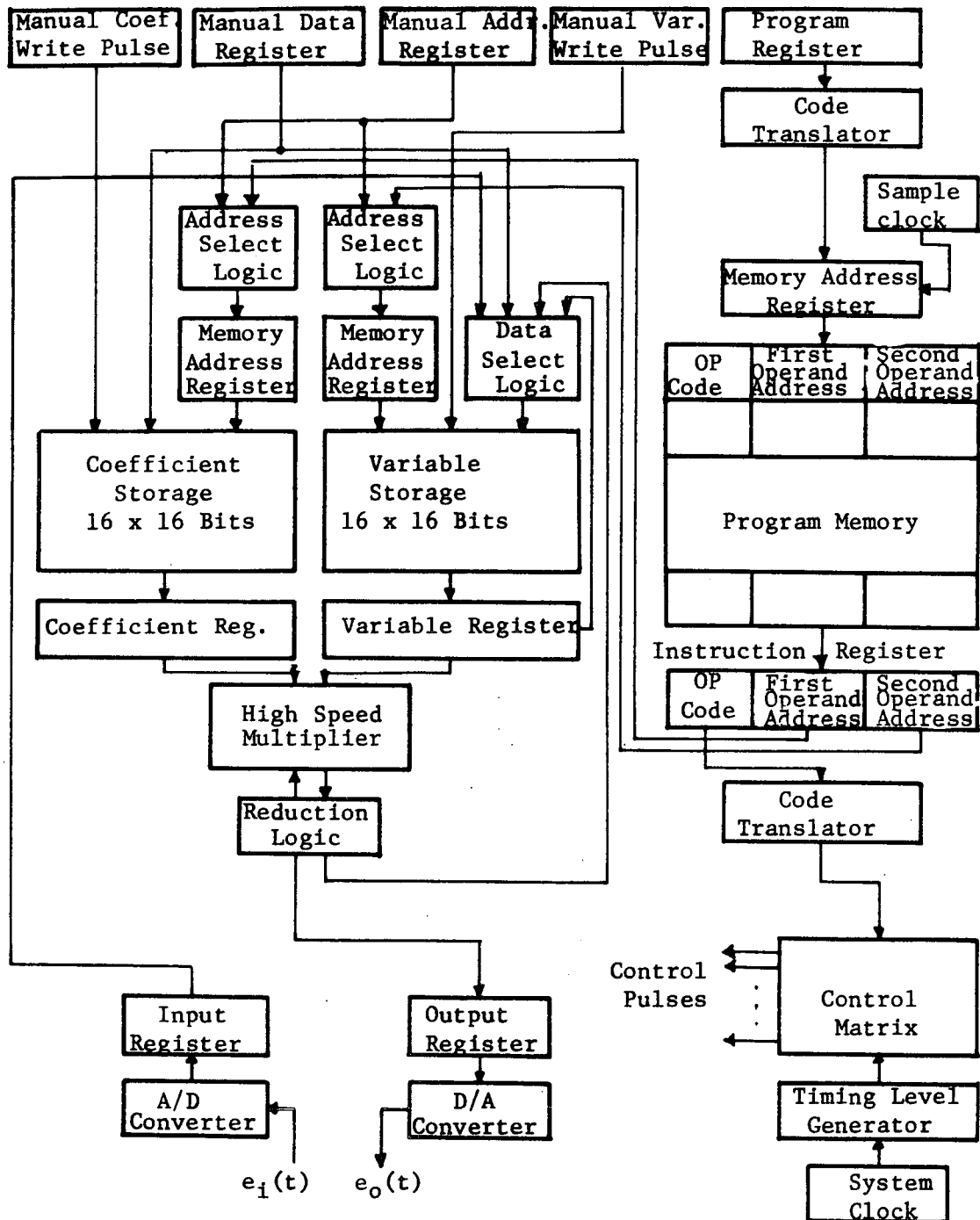


Fig. 4.19. Block diagram of a microprogrammable digital filter implementation.

of Fig. 4.7 are the A/D converter along with its input register and the D/A converter and its output register. In many cases the input register is considered as a part of the A/D converter and is not shown separately. The Memory is composed of the coefficient storage, variable storage, the program memory, and their appropriate memory address select and data select networks. The coefficient register, variable register, accumulator register, high-speed multiplier and the reduction logic network comprise the Arithmetic Unit. Included in the Control Unit are the sample clock, fundamental clock, binary counter, timing level generator, control matrix, the instruction register and its code translator. In addition to the four basic functional units which are generally used to represent digital computers, the SP computer in Fig. 4.19 employs an Operator Control Unit. The Operator Control Unit is used to program the SP computer for a particular filter form and also allows the operator to load the coefficients of the transfer function, $D(z)$, into the coefficient storage locations. The Operator Control Unit can also be used in the testing and trouble shooting of the computer.

At this point it would be desirable to present a brief description of the operation of the SP computer. For purposes of illustration, assume that a known transfer function, $D(z)$, is to be realized and that a specific filter programming form has been selected. With reference to the functional blocks of Fig. 4.19, the programming form is chosen by setting the 4 bits in the program register to the proper values

as will be defined later. Next the constants for the difference equations are manually written into the coefficient storage after switching the memory control switch, S, to the "1" position. Also, to control the output reduction logic, load the shift key words into the variable memory which are called by the quantize instruction to provide the desired number of accumulator bits to be input to the D/A or to be written into the variable storage. The programming of the SP computer is complete and it now awaits the first input signal.

Normal operation begins as the sample clock gates the translated code of the program register into the program memory address register. The memory address register contains the address of the first instruction needed to calculate the difference equations for the filter programming form chosen. Stored in the program memory in groups of consecutive addresses are the macro-instructions for all filter programming forms. For example, the first seventeen locations in the program memory are the macro-instructions for the direct programming form. Upon receiving a read pulse, the program memory loads the 16-bit instruction register. The instruction format is shown in Fig. 4.20. Its four MSB's comprise the op code (operation code), the next six bits contain the first operand address (coefficient address), and the last six bits contain the second operand address (variable address). After the op code is decoded into one of the eleven available macro-operations, the Control Unit generates a corresponding sequence of micro-operations. Thus, each macro-operation is built up as a sequence of elementary micro-operations.

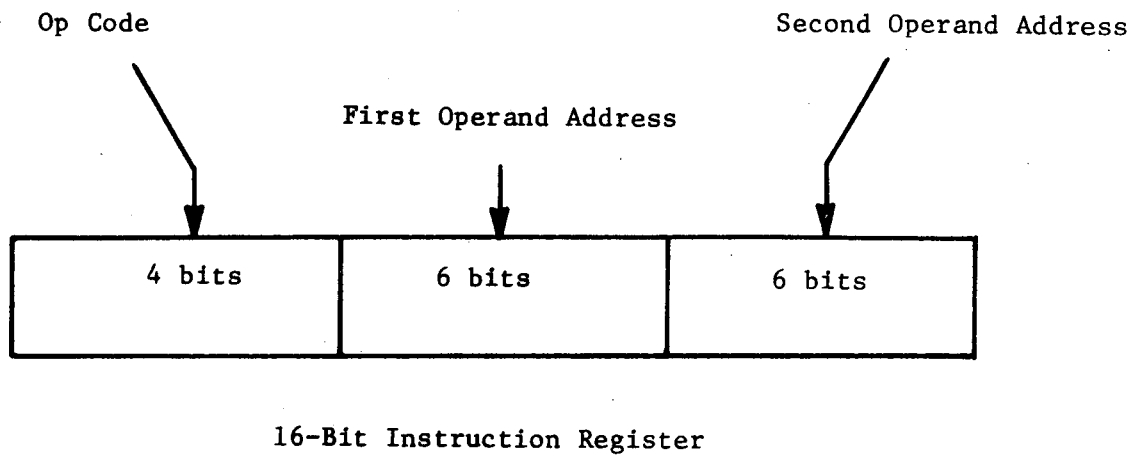


Fig. 4.20. Macro-instruction format.

The remainder of the discussion of a micro-programmable realizations will be devoted to the specification and organization of the four basic functional blocks of Fig. 4.7.

1. Input/Output Unit.

As before, it is the task of the input interface element to digitize the input analog signal $e_i(t)$ in the A/D converter and furnish this digital signal $e_i(kT)$ to the SP computer. The output, $e_0(kT)$, is in digital form and is converted to a discrete analog output, $e_0(t)$, by the output interface element, the D/A converter.

A/D Conversion. The A/D converter will be such that it may transform $e_i(t)$ into a two's complement binary representation and thus will be used as the input interface of the SP computer. Word size may vary from several bits up to sixteen bits including the sign bit. It is assumed that an A/D converter which meets the resolution and speed requirements of the SP computer is available.

A 16-bit input register is used to hold the A/D converter output. The input register functions to maintain constant values for the input data bits during the period they are used. After each conversion the A/D converter produces an end-of-conversion (EOC) signal which is used to load the new digital word into the input register. If the A/D converter wordlength is smaller than sixteen bits, the remaining bits of the input register must be filled in with the sign bit or zeroes.

D/A Conversion. Since the D/A converter is the output interface element, the selection of a D/A converter type is determined wholly by

the requirements placed on its output signal by the external system. This may lead to a D/A converter which converts a specified number of bits into either a unipolar or bipolar analog signal, the external system may require a pulse-width-modulated voltage signal, or even yet, may require a digital input, whereby a D/A converter is not needed.

For the particular micro-programmable digital filter being discussed, the D/A was chosen so that a bipolar analog voltage may be presented at the output.

2. Arithmetic Unit.

It is the purpose of the Arithmetic Unit to perform the multiplication and accumulation operations required to solve the difference equations of the digital filter programming forms described earlier. A major portion of the Arithmetic Unit consists of a high-speed multiplier. Inputs to the Arithmetic Unit are the multiplier (coefficient register), the multiplicand (variable register), and the accumulator register. Both multiplier and multiplicand inputs are 16-bits long and are coded using the two's complement representation. The output of the high-speed multiplier is a 34 bit, two's complement number which is stored in the accumulator register.

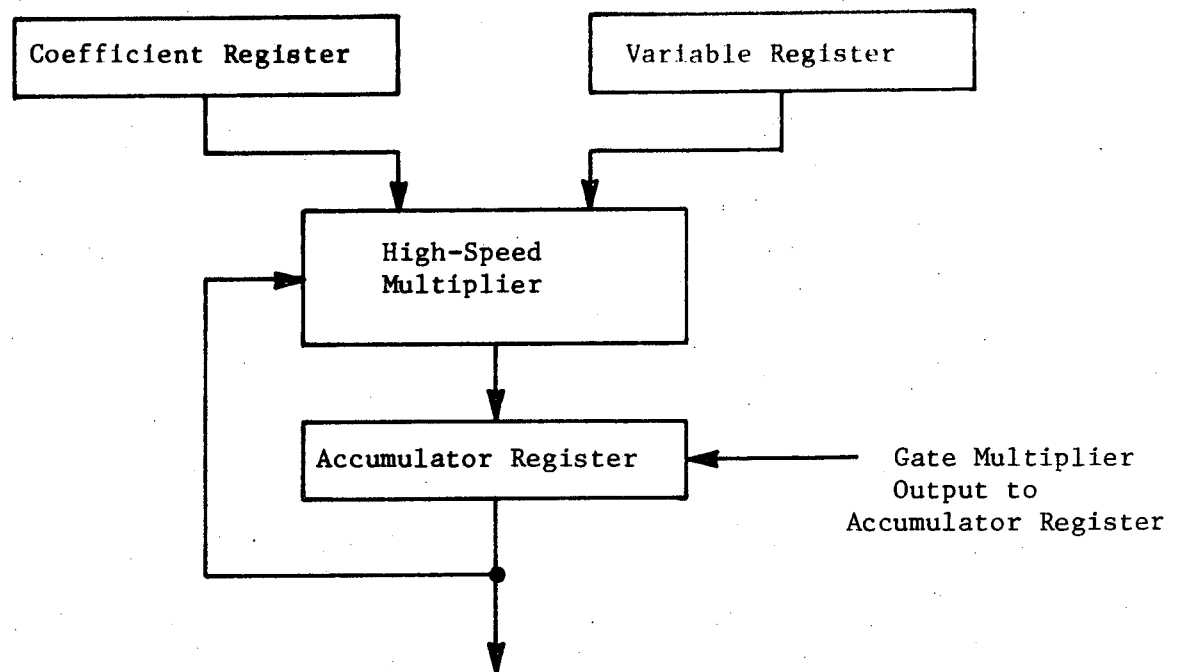
High Speed Multiplier and Accumulator Register. The data input registers (coefficient and variable) and the data output register (accumulator) are organized to permit the multiplication of the contents of the input registers and to add the results to the contents of the accumulator. The simplified block diagram of the Arithmetic Unit in

Fig. 4.21 will be used to explain the calculation of an example difference equation. Note that the input to the accumulator register is gated. This is necessary since the output of this register is fed directly into the multiplier, i.e., the next state of the accumulator register is dependent not only on the multiplier inputs but also its own present state. Thus once the output of the multiplier reaches a stable state the data is gated into the accumulator register.

The example difference equation requires two multiplications, with the results of each added to the contents of the accumulator register. Before starting the calculations, the accumulator register is cleared. Its initial contents are denoted by $(\text{Acc})_0$. Note that the accumulator register supplies an input to each column (which corresponds to a Wallace multiplier tree) of the array. This increases the size of the multiplier structure slightly but eliminates the time delay of an adder network which would otherwise be necessary to add the contents of the accumulator and the multiplier output. After the first multiplication, the result is gated into the accumulator, becoming $(\text{Acc})_1$. For accumulation of the second product, $a_1 e_1(kT - T)$, $(\text{Acc})_1$ and the partial products from the AND gate array are used as inputs to the free network. The result,

$$e_0(kT) = (12/16)(10/16) + (-10/16)(-6/16) = 180/256$$

is gated into the accumulator register after sufficient time for the inputs to propagate through the Wallace multiplier trees.



Example difference equation:

$$e_o(kT) = a_o e_i(kT) + a_1 e_i(kT - T)$$

let $a_o = 0.75 = 12/16 = 0.1100$
 $a_1 = -0.625 = -10/16 = 1.0110$

$$e_i(kT) = 0.625 = 10/16 = 0.1010$$

$$e_i(kT - T) = -0.375 = -6/16 = 1.1010$$

0.1100		1.0110	
0.1010		1.1010	
0.00000000	(Acc) ₀	0.01111000	(Acc) ₁
0.00000000		0.00000000	
0.0001100		1.1110110	
0.000000		0.000000	
0.01100		1.10110	
0.0000		0.1001	
0.01111000	(Acc) ₁ = 120/256	1	
		0.10110100	(Acc) ₂ = 180/256

Fig. 4.21. Calculation of example difference equation by Arithmetic Unit.

The first multiplication and accumulation operation is straightforward since both the multiplier and multiplicand are positive numbers. However, this is not the case for the second operation. Note, that if the product of the multiplier bit and the multiplicand sign bit is a "one", it must be repeated in the left-most positions. In the first operations, this product was "zero" in all cases and thus the left-most fill-in positions contain all zeroes. Also, note that the negative multiplier in the second operation requires that the last partial product term be the two's complement of the multiplicand. This is accomplished by taking the one's complement of the multiplicand and forcing a "1" into the Wallace tree for the LSB of this partial product. This procedure is taken care of by the partial product generation logic of the high-speed multiplier and is presented in detail in [12].

Every phase of the multiplier has been demonstrated, except the establishment of the length of the accumulator register. Multiplying two 16-bit, sign two's complement numbers yeilds a 31-bit product. Since the direct digital filter programming form requires the summation of the greatest number of products (5) in the solution of any single difference equation, this sets the required length of the accumulator register at 34 bits. This means that if two maximum size 16-bit numbers are multiplied and accumulated five times, a 34-bit register would be required to express the sum.

Reduction Logic. There are several register lengths in the SP computer which need to be analyzed; the 34-bit accumulator register, the 16-bit data locations in the variable storage, and the N-bit D/A converter. This is brought about by the use of the solution of the difference equation in later calculations or as an output, $e_o(kT)$. In the first case, this 34-bit solution must be stored in a 16-bit location. Therefore it is necessary to quantize the output of the accumulator register to 16 bits. In the second case, it is also necessary to reduce the word length, since a 34-bit D/A would be both expensive and impractical. Thus the need for reduction logic has been clearly established.

The number of bits in the output variable remaining after quantizing the contents of the accumulator register is determined by the size of the D/A converter. For an N-bit D/A converter the reduction logic may select the first N least significant bits (LSB's), the first N most significant bits (MSB's) or any intermediate group of N bits.

This selection of output data bits is accomplished by writing into the variable storage a shift key word for each quantize instruction in the program memory for a particular programming form. Part of the instructions will contain the address of this shift key, which is read from storage and gated into the reduction shift register, the contents of which determine those bits of the accumulator register to be loaded into the D/A or the variable storage. This is done by choosing the appropriate bits of the accumulator and shifting these bits to the left until they occupy those positions with output lines.

3. Memory.

Each of the sections of memory shown in Fig. 4.19 provides storage for a specific type of data. The memory, as previously discussed, is not only used for storage purposes, but is also utilized to perform the time delay operations which are required to calculate the difference equations. Included in the description of the memory sections will be the memory address registers, the address select logic and the data select logic.

Coefficient Storage. Upon selection of a transfer function and a filter programming form, it is necessary to store the proper filter constants and coefficients to be used in the solution of the difference equations. The coefficient storage is a high-speed Read/Write memory which is composed of two memory modules. Each module has addressable storage locations for sixteen 8-bit words. Proper connection of the modules yields 9 (16 x 16)-bit memory as shown in Fig. 4.22.

Data is manually written into the coefficient storage locations through the use of the data register, the memory control switch, manual address load pulse, and the manually controlled write pulse on the control panel. The address select logic in Fig. 4.22 is necessary to allow the coefficient address to be chosen from the control panel address register when manually writing coefficient values into memory or the first operand address portion of the instruction register when reading coefficient values from memory.

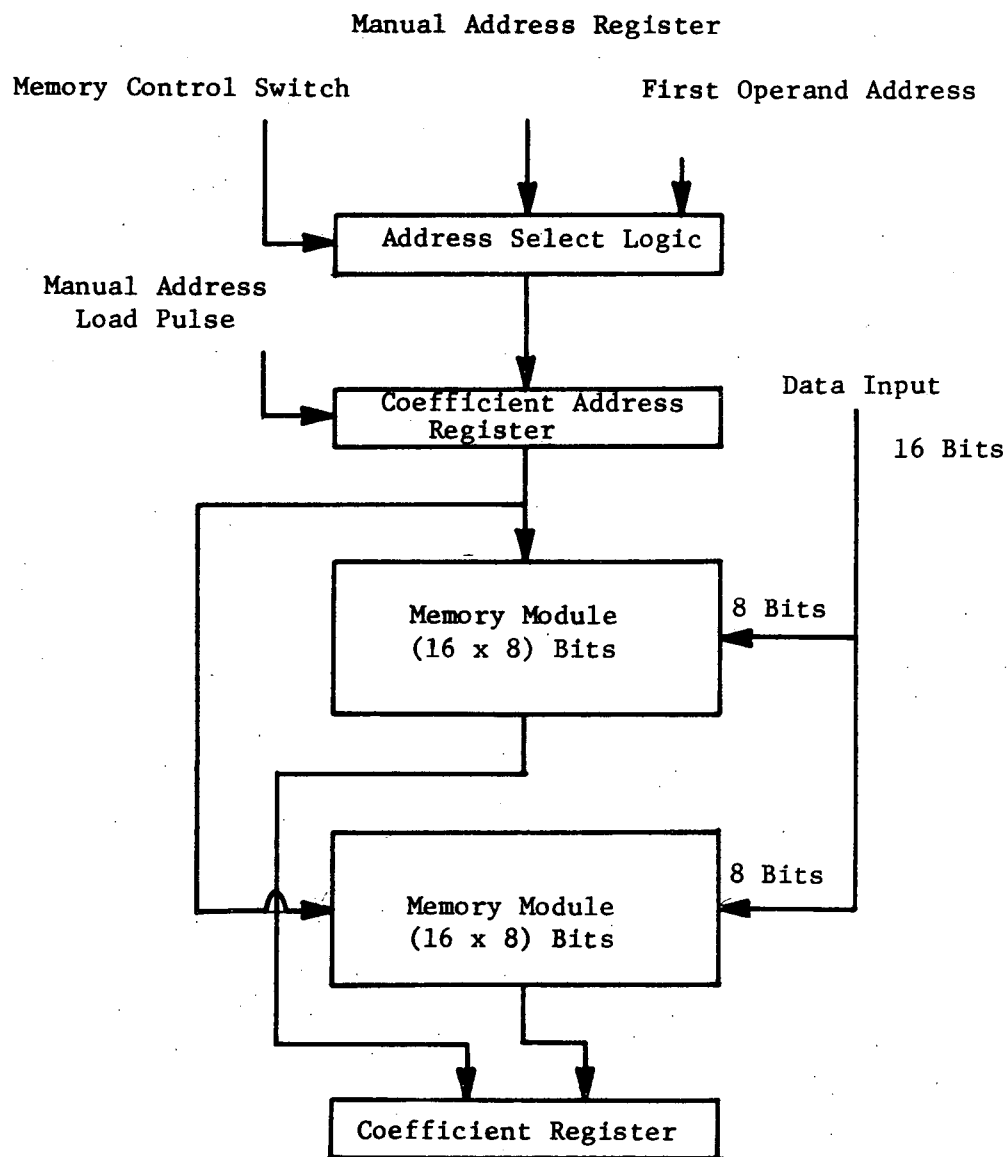


Fig. 4.22. Coefficient storage.

Variable storage. The organization of the variable storage, as shown in Fig. 4.23, is similar to that of the coefficient storage. Its function is not only that of storing the input, $e_i(kT)$, and the output, $e_o(kT)$, but also that of performing time delay. Assume the input sample, $e_i(kT)$ and its previous value, $e_i(kT - T)$, are stored in memory. After the last computation involving $e_i(kT - T)$ is completed, $e_i(kT)$ is written into the memory location allocated to $e_i(kT - T)$ where it waits until the next sampling period to be used as $e_i(kT - T)$; thus the time delay operation is performed.

Note that the variable storage requires both data and address select logic. This is due to the variable storage being used by a multiple of sinks and sources. Data inputs to the variable storage may come from either the control panel, A/D, variable register or the reduction logic. The variable storage may be addressed by the control panel or the second operand address portion of the instruction register.

Program memory. The program memory functions as a storage location for the macro-instruction necessary to solve the difference equations of the various filter programming forms. These instructions are organized into groups; each group corresponds to a particular programming form. Within each group, the macro-instructions are sequentially arranged as needed in the solution of the particular set of difference equations.

The program memory is a high-speed ROM that has 256 words of 16 bits each. Each word (macro-instruction) is broken into three sections:

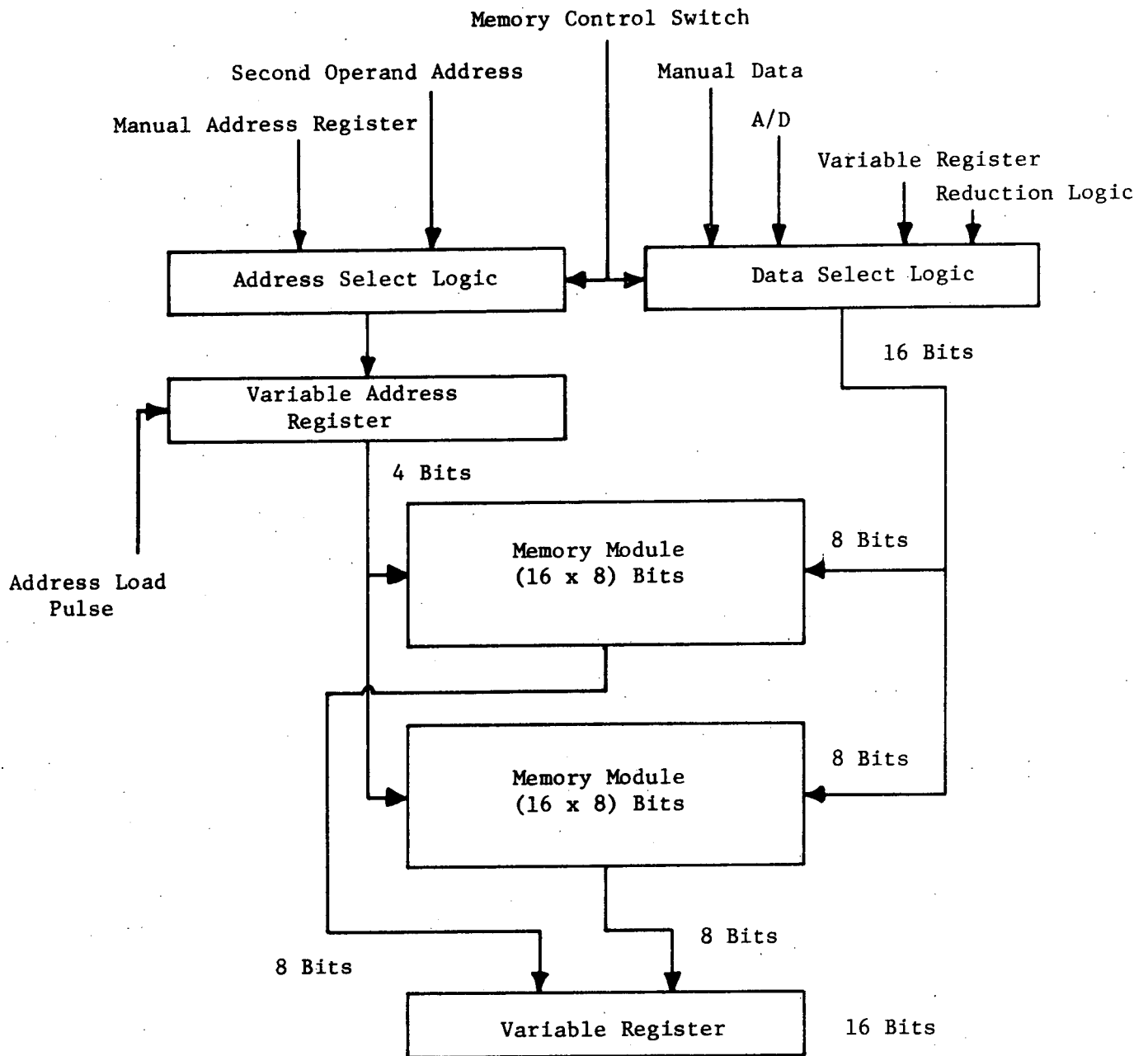


Fig. 4.23. Variable storage.

the op code (4 bits), the first operand address (6 bits) and the second operand address (6 bits). The op code specifies the operation to be performed on the operands in the memory located at the addresses specified by the two address fields. For op codes which require only one operand (or no operand), that portion of the program memory is blank, i.e., contains any combination of zeroes and ones. An example is the store input instruction. Here the op code is "0101", the coefficient address is blank and the variable address contains a 6-bit code specifying the storage location in the variable storage which is to receive the input data.

Since the program memory contains groups of macro-instructions for all filter programming forms, it is necessary to be able to locate the first address in each group once a form has been chosen. The contents of the program register is translated into the program memory address of the first macro-instruction for each filter programming form. Next, the sample clock gates the translated code into the program memory address register (PMAR) and the first macro-instruction is accessed by a memory read pulse. First and last in every sequence of macro-instructions is a start A/D instruction. This is necessitated by the overlapping of the instruction and execution cycle.

In order to program the SP computer, it is necessary to code each digital filter form using 4 bits. Table 4.1 presents the coding scheme for the filter being described in [12].

TABLE 4.1. Program Code

Program Register Contents	Digital Filter Programming Form
0000	Direct
0001	Modified Direct
0010	Standard
0011	Modified Standard
0100	Canonical
0101	Modified Canonical
0110	Parallel
0111	Cascade
1000	Modified Cascade
1001	Structure XI
1010	Structure XII
1111	Test Mode

C

As an example of the sequence of macro-instructions in the program memory, consider the direct programming form with the difference equation,

$$e_0(kT) = a_0 e_i(kT) + a_1 e_i(kT - T) + a_2 e_i(kT - 2T) \\ - b_1 e_0(kT - T) - b_2 e_0(kT - 2T) \quad (4-18)$$

Table 4.2 presents a word description of the macro-programming instructions for this form.

Notice that during the time the A/D is converting the analog voltage signal to a digital signal the computer is calculating the portions of the difference equations that do not require the digitized input, $e_i(kT)$. This is intended to minimize the time delay between the input sample and its output response.

The macro-instruction "Store (variable)" means load the contents (where "contents" is denoted by the enclosing parentheses) of the variable register into the specified address of the variable storage. Store (reduction logic) means to write the output of the reduction logic into the variable storage. In order to permit the use of A/D converters with different conversion rates, a "wait on A/D" instruction is incorporated. If the conversion is complete at the time that this instruction is reached, the next instruction "Store input" is executed; otherwise, the computer idles until it receives the end of conversion signal from the A/D. However, there may be instances when a high-speed

TABLE 4.2. Program Memory Contents for Direct Form

Op Code	First Operand Address	Second Operand Address
Start A/D		
Clear Accumulator		
Multiply & Accumulate	a_2	$e_i(kT - 2T)$
Multiply & Accumulate	a_1	$e_i(kT - T)$
Store (variable)		$e_i(kT - 2T)$
Multiply & Accumulate	$-b_2$	$e_0(kT - 2T)$
Multiply & Accumulate	$-b_1$	$e_0(kT - T)$
Store (variable)		$e_0(kT - 2T)$
Wait on A/D		
Store Input		$e_i(kT - T)$
Multiply & Accumulate	a_0	$e_i(kT - T)$
Quantize (Acc)		Shift Key
Store (Reduction Logic)		$e_0(kT - T)$
Quantize (Acc)		Shift Key
Load D/A		
Reset RMAR & Halt		
Start A/D		

A/D converter is employed. In this case the input $e_1(kT)$ is stored and not utilized in the calculations until all other multiplications are performed. This may also be an unnecessary delay in the response of the filter. Thus, it may be appropriate to postpone the "Start A/D" instruction in the program to ensure that the "Wait on A/D" instruction places the computer in an idle state.

The codes for the macro-instructions are listed in Table 4.3. This table is used to generate Table 4.4, which presents the actual contents of the program memory, the variable storage and coefficient storage for the direct filter programming form. Addresses for the program memory are encoded in octal in Table 4.4.

Note that in Table 4.4 the first and second operand address fields are 6-bits, while the actual memories (coefficient storage and variable storage) have only 4-bit addresses. Thus after loading the instruction register only the four right-most bits of each of the address fields are used as addresses for reading the contents of the coefficient and variable storages. Notice, also, the shift key word in the variable storage at location, 0100. This shift key is requested by the quantize op code, 0111, and is loaded into the reduction shift register. A shift key word exists for each quantize instruction.

Space is allocated in the program memory for the remaining programming forms. They will not be illustrated since one may get an idea of their structure from observing the direct programming example.

TABLE 4.3. Macro-Instruction Op Codes

Op Code	Operation
0000	Start A/D
0001	Clear Accumulator
0010	Multiply & Accumulate
0011	Store (variable)
0100	Wait on A/D
0101	Store Input
0110	Load D/A
0111	Quantize (Accumulator)
1000	Store (Reduction Logic)
1001	Halt
1010	Reset PMAR and Halt
.	
.	
. Not Used	
.	
.	
.	
1111	

TABLE 4.4. Memory Contents for Direct Form.

Program Memory			
(Address) ₈	Op Code	First Operand Address	Second Operand Address
000	0000		
001	0001		
002	0010	000000	000000
003	0010	000001	000001
004	0011		000000
005	0010	000010	000010
006	0010	000011	000011
007	0011		000010
010	0100		
011	0101		000001
012	0010	000100	000001
013	0111		000100
014	1000		000011
015	0111		000101
016	0110		
017	1010		
020	0000		

Address	Coefficient Storage	Address	Variable Storage
0000	a_2	0000	$e_1(kT - 2T)$
0001	a_1	0001	$e_1(kT - T)$
0010	$-b_2$	0010	$e_0(kT - 2T)$
0011	$-b_1$	0011	$e_0(kT - T)$
0100	a_0	0100	Shift Key
0101		0101	Shift Key

Program Memory Address Register. Macro-instructions retrieved from the program memory are read from memory addresses contained in the program memory address register (PMAR). Inputs to PMAR came from the code translator and the control unit. Data from the code translator is loaded into PMAR by the sample clock, whereas, upon receiving the control pulse, "up date PMAR", PMAR performs the operation

$$(PMAR) + 1 \rightarrow PMAR,$$

and now contains the address for the next macro-instruction. Since there are 240 addresses in the program memory, the PMAR must be eight bits long.

4. Control Unit

The program for a digital computer consists of a set of machine operations such as addition and multiplication. Instructions for these operations have been referred to as macro-operations. Inside the SP computer these operations are further decomposed into a set of elementary operations called micro-operations. Count, shift, gate the memory address register, are examples of micro-operations. Normally, in general purpose computers, macro-instructions are at the programmer's disposal and may be readily changed by re-writing the program. However, as previously described, in this SP computer the macro-instructions are pre-programmed in the program memory. These instructions are sequentially read from the program memory and loaded into the instruction register. In the op code portion of the instruction register is a 4-bit code

which specified the macro-instruction to be executed. This op code is fed to the control unit of the SP computer and a sequence of micro-operations is performed for each op code.

It is the purpose of the Control Unit to translate the op codes and supply all synchronization and control pulses to the rest of the SP computer. A block diagram of the Control Unit elements is shown in Fig. 4.24. These elements include the instruction register, a decoder, a four-state counter, a timing level generator, a control matrix, a fundamental clock and an operation flip-flop, D. The organization and function of each of these elements is discussed below.

Instruction register and decoder. As macro-instructions are retrieved from the program memory, they are stored in the instruction register until the instruction is executed and a new one is retrieved. Only the op code portion of the instruction register is employed by the rest of the Control Unit, while the operand addresses are sent to the coefficient and variable memory address registers.

There are eleven macro-instructions which are used by the SP computer to solve the difference equations of the various digital filter programming forms. Each op code portion of a macro-instruction is translated by the decoder into one of the macro-operations, f_i , $i = 0, 1, 2, \dots, 10$. For each op code, the decoder activates one and only one output line.

Fundamental clock. Employed as a fundamental clock is a free-running multivibrator whose frequency is obtained after determining a basic timing cycle for the SP computer, which can be done after the

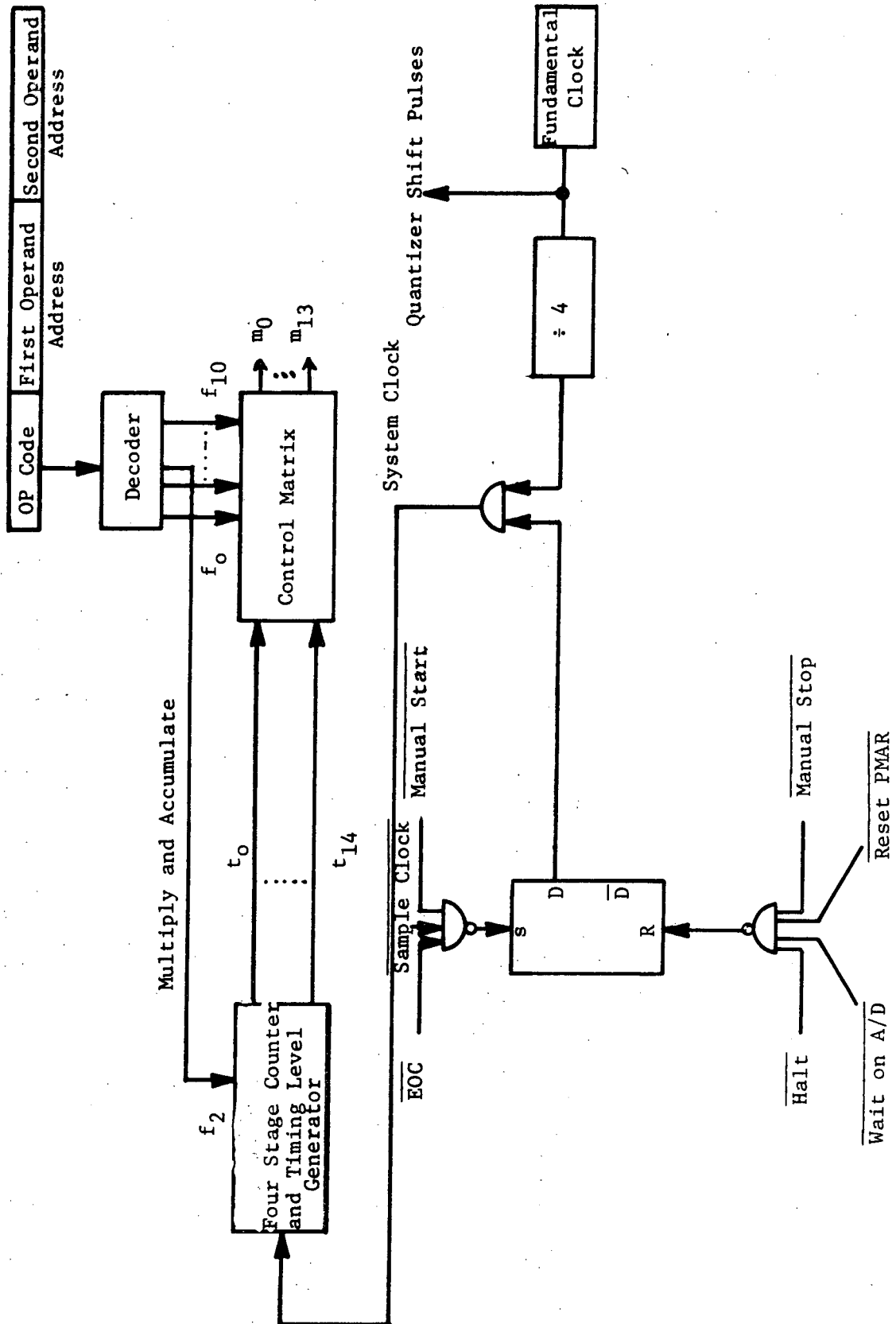


Fig. 4.24. Control unit.

detailed logic design of each functional unit is complete. Note that a clock signal that is $1/4$ as fast as the fundamental clock is used as input to the Control Unit. This is because one of the micro-operations, quantize (Acc), is a serial operation and requires these high frequency pulses in order to avoid slowing the operation of the SP computer.

Four stage counter and timing level generator. There are fourteen micro-operations (from which eleven sequences of micro-operations are formed for the eleven macro-operations), and the control signals for these micro-operations are designated as m_j , $j = 0, 1, \dots, 13$. The number of timing levels for the execution of the macro-instructions will require six states of the four stage counter, L , for all instructions except multiply and accumulate, whose last micro-operation is performed on state fourteen. Thus this macro-instruction is used as a control input to the counter.

Control matrix. Basically, the function of the control matrix is to provide the proper combination of op codes and timing levels. Essentially, the control matrix is an AND-OR switching matrix.

Operation flip-flop. Controlling the operation of the SP computer is the operation flip-flop, D . If D is set, clock pulses enter the four stage counter under normal operations. Flip-flop D may be set by three signals, the manual start, the end-of-conversion (EOC) signal from the A/D and the sample clock pulse. It may be reset by the control pulses, wait on A/D, halt, and reset PMAR and the manual stop button. Note that when the computer is halted by reset PMAR, the first instruction

to be executed is already in the N register ready for execution. Care must be taken to ensure that EOC does not occur before the wait on A/D signal during the execution of a given filter programming form.

5. Operator Control Unit.

As mentioned earlier, the Operator Control Unit is used in manually programming the SP computer to realize the transfer function, $D(z)$, in the selected digital filter programming form. Referring to Fig. 4.19, the elements comprising the Operator Control Unit are the manual data register, the manual memory address register, two write pulse circuits (one each for the coefficient and variable storage), a program register, a memory control switch, a manual start pulse circuit, a manual stop pulse circuit, two pulse circuits for manually loading the coefficient and variable memory address registers, a row of sixteen indicator lights and a monitor switch, and a sample clock enable switch.

Each of the registers may be implemented with single pole-double-throw (SPDT) switches for ease of setting and resetting.

Data may be entered into either the coefficient or variable storage by setting the manual memory control switch to the "1" position, setting the switches of the manual data register to the binary data value, setting the address into the manual address register, loading this value into the coefficient or variable memory address register, and pressing the coefficient or variable write button. Notice the data register and memory address register are common to both the variable and coefficient storage since each employs a separate load address and write pulse.

Previous discussion has described the functions of the program register, manual start button and the manual stop button. The indicator lights and monitor switch function in checking the operational status of the SP computer.

From the above discussion, one should understand the basics of a microprogrammable digital filter implementation. If further details are desired, one may refer to [12].

Next to be discussed will be the aspect of timesharing digital filter implementations.

Time-Sharing of a Digital Filter Implementation

For many applications of a digital filter it is sometimes necessary to provide discrete-time filtering for several independent loops or channels with examples being in control systems and digital communications systems. This may be accomplished with several digital filters but it would be more practical, more reliable, and more economical to provide this filtering with a special-purpose computer organized and programmed as a time-shared digital filter [9].

There are two basic ways in which a digital filter may be time-shared. The first of these is in multi-loops and the second is for higher order $D(z)$ realization.

Fig. 4.25 illustrates a digital filter being time-shared in N control loops. It is seen that there is an enormous amount of hardware saved by doing this. First there is only one computing element (SP

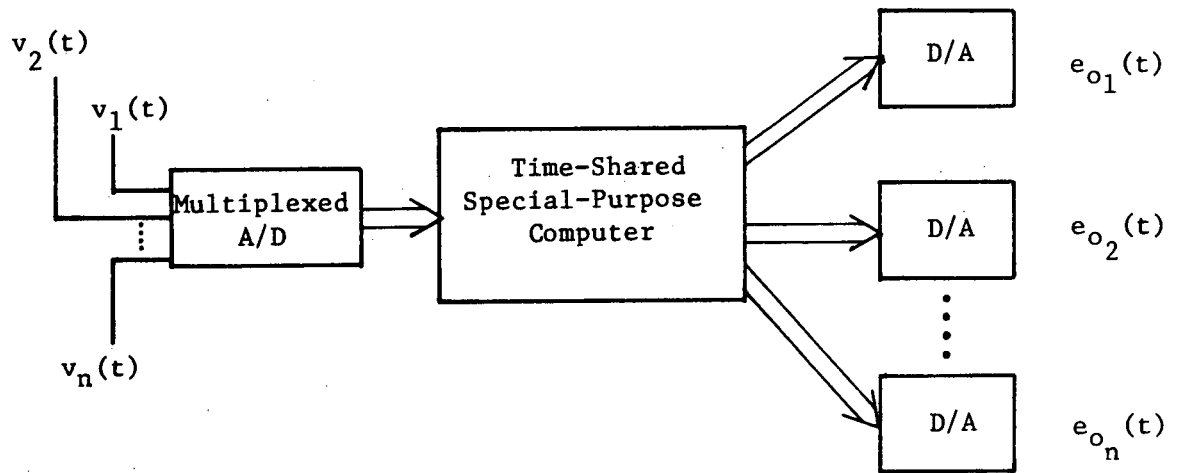
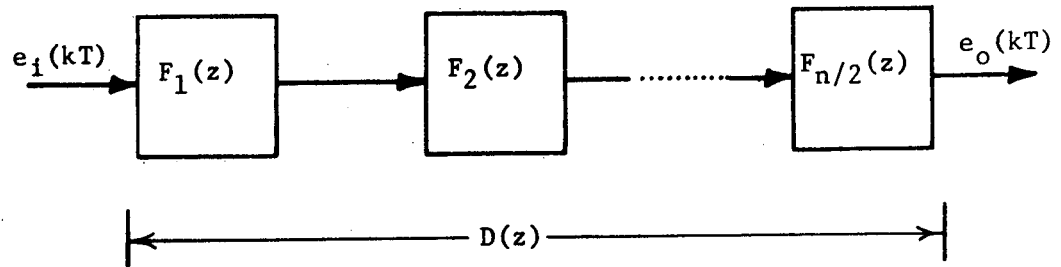


Fig. 4.25. Block diagram of a digital filter being time-shared in multiple loops.

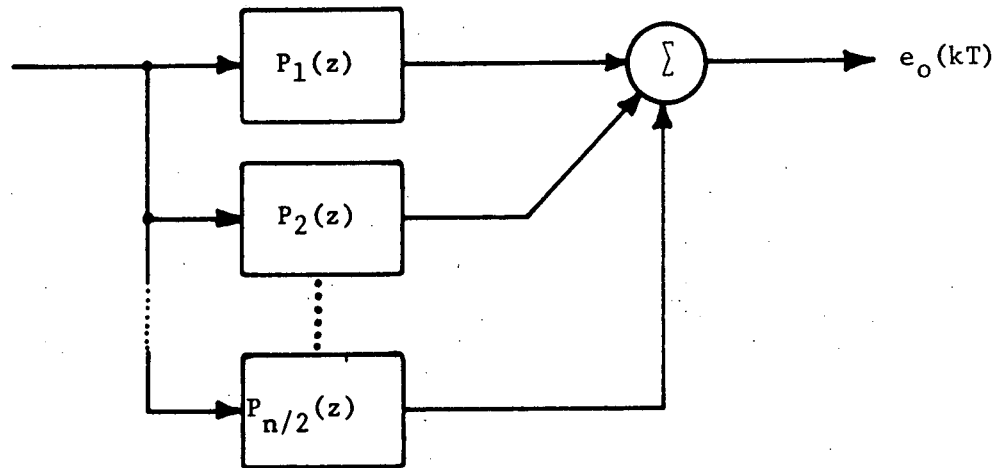
computer) required and second only one A/D is required, whereas if time-sharing weren't used N computing elements and N A/D converters would have been required.

Fig. 4.26 illustrates a digital filter being time shared for a higher order $D(z)$ realization. This can be done by cascade or parallel methods as shown in parts (a) and (b) of the figure. Cascading or paralleling filters to obtain a higher order realization is preferred because of the word length problem (resolution) that is encountered for a single high order $D(z)$ realization. Time sharing of a single filter as shown in Fig. 4.26 can be easily accomplished if the assumption is made that n is even and that $D(z)$ can be factored into $n/2$ second-order filters $F_1(z), \dots, F_{n/2}(z)$ and partial fractional into $n/2$ second-order filters $P_1(z), P_2(z), \dots, P_{n/2}(z)$. Each of the second-order filters may be realized by any of the available programming forms. The total transfer function $D(z)$ still has the same memory requirements, the same number of memory locations in each filter storage unit, but now the variables and coefficients of each second-order filter are the signals stored. Since each second-order filter has the same program, the control sequence generator just repeats the same sequence $n/2$ times during each sampling interval.

Both cascade and parallel techniques are desirable methods for realizing higher order $D(z)$'s, but the additional summing junction necessary with the parallel method makes the cascade method better suited for a modular realization.



(a)



(b)

Fig. 4.26. (a) Cascade and (b) parallel methods of realizing a higher order $D(z)$.

C5

For a better understanding of time-shared digital filter implementations, let us look at the organization of a SP computer realization of a 2nd order time shared digital filter which may be time shared in two loops or cascaded or paralleled for the realization of a 4th order $D(z)$.

Fig. 4.27 is a functional block diagram of the SP computer realization of two digital filters. Like previous SP computer realizations, the computer functions are still divided into four main categories; input-output equipment, memory unit, arithmetic unit, and control unit.

The input-output equipment provides the interface between the analog system and the digital computer. The analog inputs are sampled via the multiplexer and A/D, and the output samples are demultiplexed and shared in the buffered D/A's.

The arithmetic unit, for this type realization, also must be able to perform multiplication, addition and subtraction. The selection of an arithmetic unit must first entail the selection of serial or parallel type arithmetic operation. Usually this choice means a decision must be made between computational speed and hardware economy. The choice made here was a reasonable fast and economical method where a parallel binary accumulator and two shift registers are used to add up the product terms as previously discussed. Serial arithmetic is used for partial product multiplication in the implementation discussed in [18].

As before, the reduction logic truncates to maintain word-length compatibility, and saturation logic sets the output word to maximum value when its range is exceeded.

The memory is divided into two identical storage units for filter-1 storage and filter-2 storage. Corresponding filter signals such as $e_1(k - 1)$ (short notation for $e_1(kT - T)$) and $e_2(k - 1)$ have the same relative storage locations in their respective storage units and have the same addresses within their respective modules. The memory is controlled by Read and Write commands (not shown) and two address commands: a filter address to select the proper storage unit, and a signal address to select the proper signal location in a storage unit. The filter selection logic is controlled by the filter address and determines which storage unit is addressed by the signal address. This modular arrangement of the memory is ideally suited for the "wired-OR" feature of integrated circuit memory. With this, feature storage for additional filters can be added by hard-wiring inputs and outputs of the storage units and simple modification of addressing.

The control unit contains the master control unit that provides the multiplexing by means of the filter address and determines the sampling rate for each filter by controlling the start of each difference-equation computation. The control unit also contains the control-sequence generator which provides a sequence of instructions, initiated by a pulse from the master controller, to control the difference-equation computations. Thus the hard-wired program of the control-sequence generator determines the programming forms of the filters. To maintain simplicity, the same programming form is chosen for each filter, and hence the same sequence is generated each time regardless of which filter

Reproduced from
best available copy.

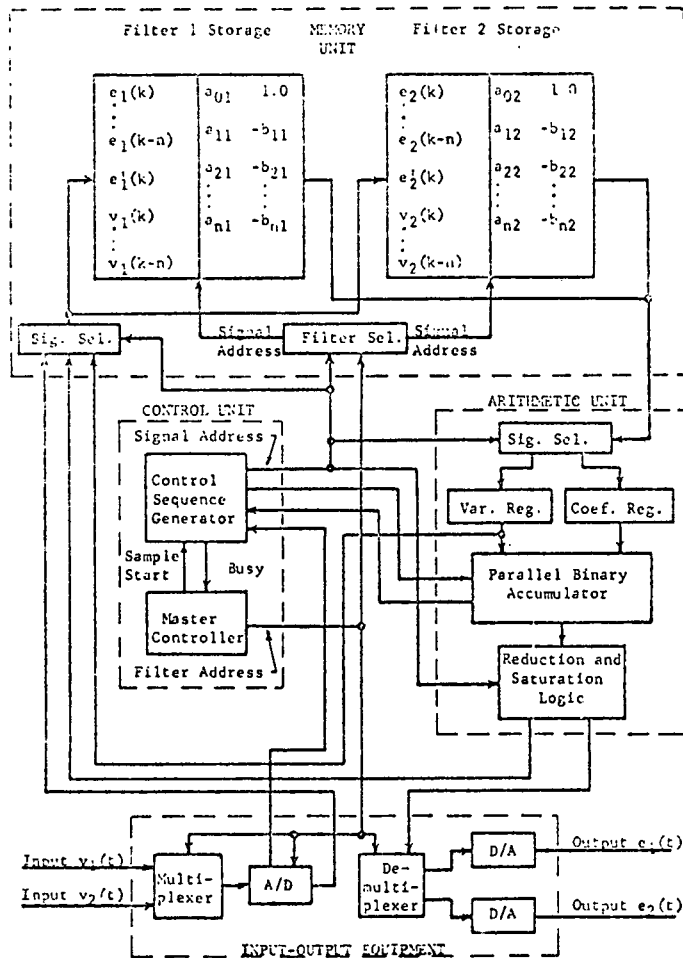


Fig. 4.27. Block diagram of the time-shared realization of two digital filters.

is being addressed and regardless of the number of filters being realized. A priority system is provided so that once the computation of a difference equation has begun, it is carried to completion even if it means that a sample for another filter must be omitted.

Range Switching Digital Filter Implementation

For many applications of digital filtering, it is desirable to have a very fast sampling rate. An example of this is when very high frequency signals are being filtered, and since the sample rate must be at least twice the highest signal frequency, it is seen that very high rates can be required. The limiting factors for a filter's sampling rate are the conversion speed of the A/D and D/A input-output system and the time required in the arithmetic calculations of the difference equations being realized. The most important factor which limits the speed of arithmetic calculations is the bit-lengths of the data processed internally by the filter. This includes the length of the input word to the filter, the internal variable wordlength (length of $e_i(kT - nT)$, $m(kT - nT)$, etc.) and the output wordlength. In general, the shorter these wordlengths are the faster the arithmetic calculations can be performed, with this being true for serial and parallel arithmetic type filters. Also, as a result of shorter wordlengths, the filter hardware is reduced.

Because of the quantization error introduced by shorter wordlengths, a scheme must be devised to eliminate the larger quantization errors

introduced by the shortened words. A method was devised to do this and the resulting filter was called a "Range-Switching" digital filter as described in [8]. In short, the filter with reduced wordlength performed as if it had a much longer wordlength, under certain conditions.

A block diagram of the filter described in [8] is shown in Fig.

4.28. Its operation will now be described.

The A/D converter converts the analog input signal into an 8-bit digital approximation allowing the digital output of the converter to have an integer value ranging from 0 to 255. The input select logic selects either the four MSB's or the four LSB's of the A/D output to be the input to the SP computer. The scheme used is to select the four LSB's if all the four MSB's are logic "0" or the four MSB's if either one is a logic "1" for a signed magnitude code. It is seen from this that the input to the filter is in either of one or two ranges. If the input to the SP computer is from the lower range (four LSB's), it may have any integer value between 0 and 15 in steps of 1. If the input is in the higher range (four MSB's) it may have any integer value between 16 and 255 in steps of 16. Of the two ranges, it is seen that the higher range has the larger quantization step h and it is 16 times that of the lower quantization step. This also means that a four bit combination coming from the higher range input would represent a magnitude 16 times the same bit combination coming from the lower range input as shown in Fig. 4.29. Because of this, before the special purpose computer can process its four bit input each sample period, it must know the range from which it comes to properly reweight

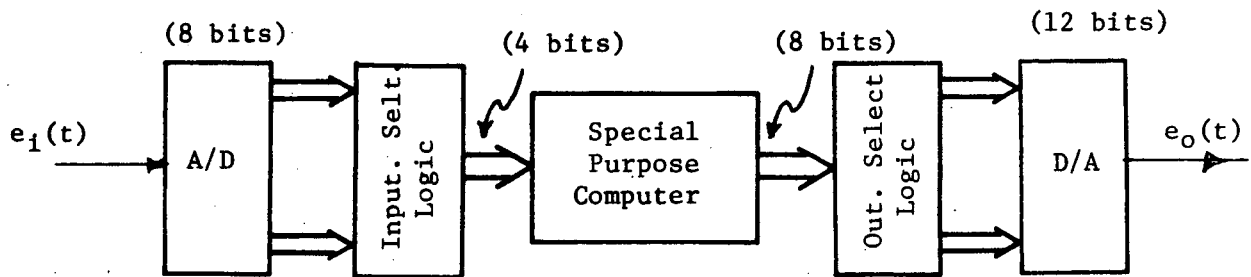


Fig. 4.28. Block diagram of a "range-switching" digital filter.

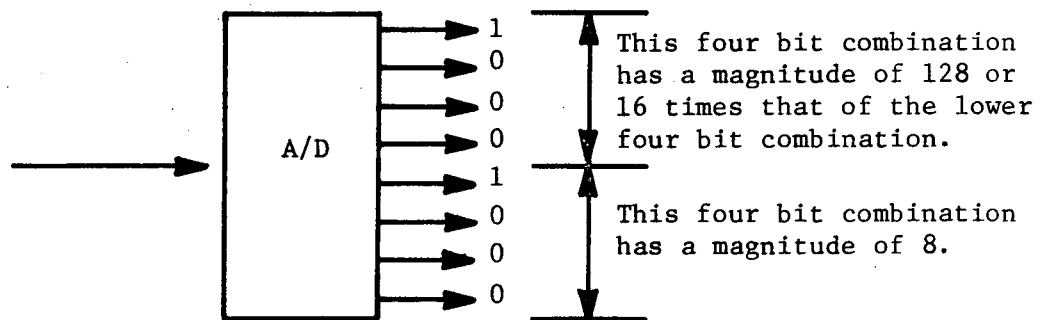


Fig. 4.29. The two magnitudes ranges.

the internal variables if a range change is seen from the last sample period. The internal variables of the SP computer are 8-bits long and can possibly be multiplied or divided by a factor of 16 each sample period before calculation of the difference equations start or their weight may remain the same. If there is a change in the input from the higher to lower range the internal variables are multiplied by a factor of 16 (the relative magnitude change of the input). It must be remembered that this is being done to keep the weight of the input relative to the internal variables. If the input represents a small value, the internal variables must be increased to make the input appear small. The opposite of this takes place when there is a change from the lower to the higher range. In this case the internal variables must be made to appear small to the large input, so they are divided by a factor of 16. If there is no range change between sample periods, the weight of the internal variables remains the same. The output select logic in Fig. 4.28 is employed to properly weight the 12-bit output of the filter. The output will have its greatest weight when the input for a particular sample period was in the higher range. If the input for a particular sample period is in the lower range, the output select logic weights the output bit configuration such that its analog voltage level representation is 1/16 of that of the same bit configuration with the input in the higher range.

It was mentioned earlier that the "range-switching" scheme was used to reduce wordlengths but at the same time obtain the accuracy at

the filter output of a much longer wordlength filter. Also, the "range-switching" filter might be thought of as a technique by which for a fixed wordlength input, the quantization errors are reduced by the range-switching process.

Filters of the "range-switching" design seem to have a bright future, especially with the advent of LSI. Using a reduced wordlength modular filter design, it would be very easy to have a digital filter composed of four or five LSI chips.

One application of "range-switching" filters that has great promise is that of filtering in nulling type control loops. First, the design saves hardware because of the reduced wordlengths, it is fast and because of the design of the filter, the quantization step length decreases as the loop error is nulled toward zero giving the loop finer granularity control to keep the error signal closer to zero.

A block diagram illustrating the components of the SP computer realization of the "range-switching" digital filter is shown in Fig. 4.30.

The input/output unit consists of a successive approximation type A/D converter, a 12-bit D/A converter and the input/output select logic which are combinational circuits that select the input word and insert the output word into the proper bit position of the D/A. As previously mentioned the output of the filter is an 8-bit word. The 12-bit D/A converter is required such that the 8 output bits can be inserted into the 8 MSB positions of the D/A when the input is in the higher range

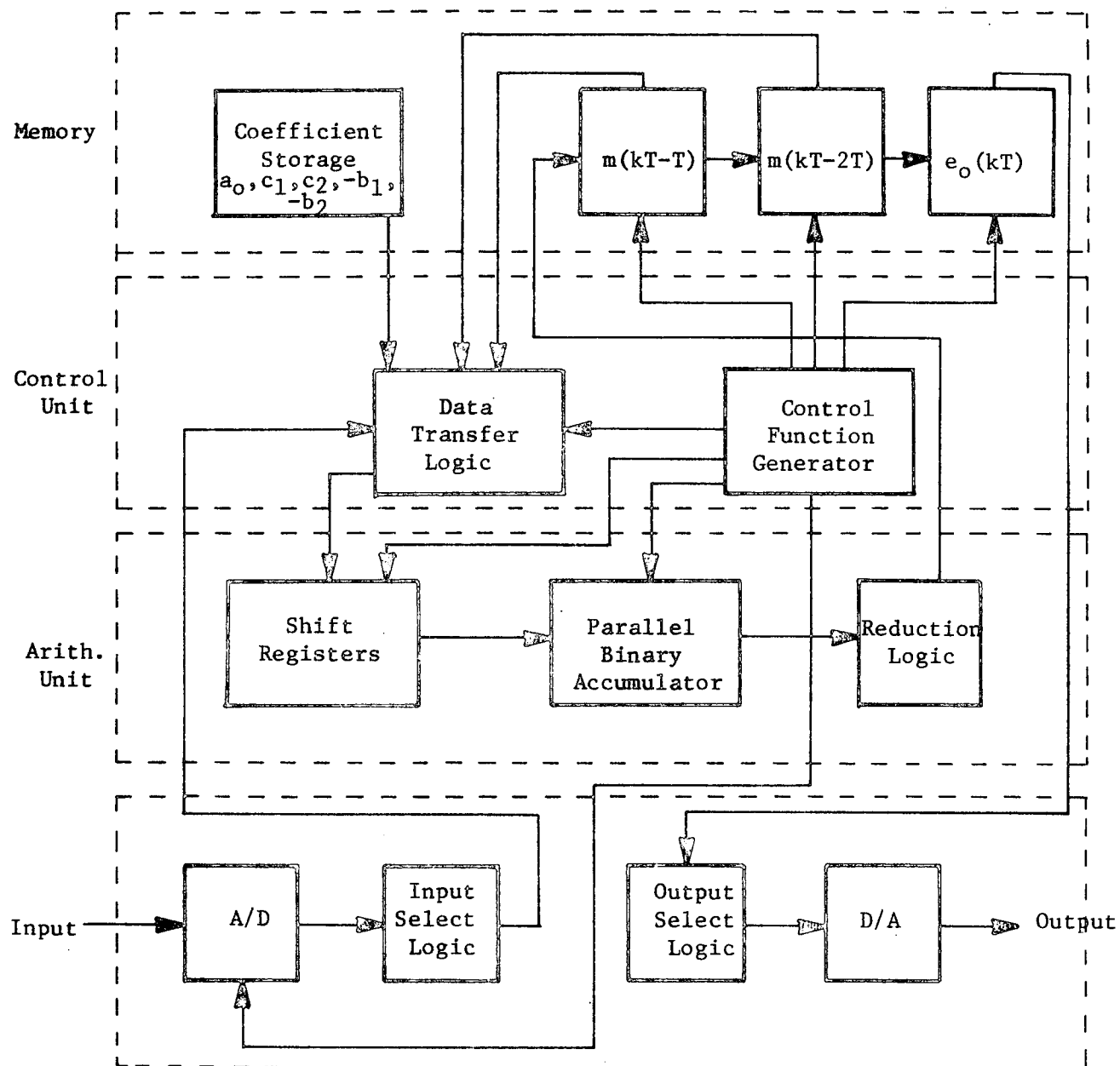


Fig. 4.30. Functional diagram of a range adaptive filter.

and in the lower 8-bit positions of the D/A when the input of the filter is in the lower range. In effect, a particular bit combination that is inserted into the lower 8-bit positions will have an analog voltage level $1/16$ of what it would have been if inserted into the 8 MSB positions. It should be noted that the weighting factor of the output is the same as the input.

Parallel fixed point arithmetic is used in the arithmetic unit in conjunction with shift registers, an accumulator, and the reduction logic to calculate the difference equations.

The memory is composed of the coefficient storage (SPDT Switches), internal variable storage (flip-flop registers), and the output storage (flip-flop register). The weighting of the internal variable memory is under command of the controller. Since fixed point arithmetic is used, multiplication or division by 16 is easily accomplished by shifting the internal variables left or right respectively four places relative to the binary point. If a weighted bit is lost when left shifting, provisions are made for the 8-bit internal variable to be saturated (all one's for a signed magnitude code).

In concluding the discussion of the hardware implementation, the control unit is composed of the control function generator and the data transfer logic. In addition to the normal tasks of the control function generator, it has the duty of deciding how the internal variables must be weighted each sample period.

A common question arises as to what is the limit on bit-length reduction. The most general answer to this is it depends on the application of the filter. The method of determining the minimum bit length is the trial and error technique of simulating the filter in whatever configuration it is to be used. As an example, the above described "range-switching" filter was used in a pendulous integrating gyroscopic accelerometer control loop. Time simulations in FORTRAN of the loop with the filter inserted demonstrated that the loop could be stabilized for pulse inputs to the loop with the filter having a 4 magnitude bit input, 6 magnitude bit internal variables and a 8 magnitude bit output. From these bit lengths it is seen that an LSI realization of the filter would be quite small and simple.

At the present time further work is being done to investigate the possibility of further bit length and hardware reductions of a digital filter such that LSI implementations will even be more attractive.

LSI Circuit Digital Filter Implementation

The first complete LSI implementation of a digital filter was designed and built by Autonetics Division of North American Rockwell, Anaheim, California. Concerning the state-of-the-art of digital filter implementation techniques, the design was the ultimate. The design is completely modular and therefore it is easily adaptable for an LSI realization. There are 2 main chips, a serial-parallel multiplier chip and a shift register chip.

The serial-parallel multiplier chip is arranged such that it can perform all arithmetic functions required for a digital filter implementation: addition, subtraction, and multiplication. The shift register chip contains shift register memories for the internal variables, input word, output word and also the control circuitry of the filter.

The interface elements of the LSI implementation are external to the filter. The A/D output is fed into the filter in a serial manner and the filter output is also serial. The binary code used by the filter is two's complement.

The internal wordlengths of the filter are adjustable. To change them, all one has to do is to make connection changes on the chips. The filter as designed has fixed length coefficients and each being 16-bits long. They are easily set by single pole double throw switches.

The filter realizes any first, second, or third order $D(z)$ which has real poles in the parallel programming form.

Commercial Digital Filters

Now that several digital filter implementation techniques have been presented, a short discussion of digital filters available on the commercial market will be in order. Because of the relative newness of the area and the recent advent of LSI circuitry, there are few commercial builders around, two of which are listed below.

One of the first builders of a digital filter for the commercial market was the Rockland Systems Corporation of Blauvelt, New York. They

now produce a line of programmable recursive digital filters which meet a wide variety of signal processing requirements. All of their filters are composed of four basic components - adders, multipliers, shift-register delays, and memory with a modular approach being adapted to provide the greatest possible flexibility and efficiency as is described in [18,19]. The basic components are usually combined into second-order building blocks (two poles and/or two zeroes) and these blocks are then combined or multiplexed to realize any number of filters of any desired order. Programmability is achieved by employing a read/write coefficient memory. Fixed filter characteristics may be obtained with a read-only memory. Rockland produces a series of filters designed in the above manner.

Rockland also produces a programmable tenth-order recursive digital filter which can realize arbitrary "all pole" designs such as Butterworth, Bessel, or Chebyshev low-pass, high-pass, or band-pass filters. Up to 10 pole positions can be programmed through ten 12-bit filter coefficients; while up to 10 zero's can be positioned at DC or the Nyquist frequency, or can be deleted altogether. Sampling rates of up to 100 KHz can be achieved.

Electronic Communications, Inc. (ECI) is the second commercial builder of digital filters to be discussed [26].

The filters produced by ECI are the nonrecursive type and they are actually signal-processing instruments that perform the convolution integral in order to effect a filtering function. This means the filters

work strictly in the time domain and is represented by its impulse response and not by its amplitude and phase characteristics as are some nonrecursive filters. The analog input signal to this filter must be band-limited, and is accomplished by using a low-pass analog prefilter. The input bandwidth is restricted to less than half the sampling rate. One of their more common filters with a sample rate of 10 KHz has a pre-amplifier cut-off at 2.5 KHz.

The ECI digital filter has two identical shift register memories. One stores samples of the band-limited input signal, while the other contains samples of the impulse function of the filter that is desired. The sampled impulse response is represented by coefficients that are the various amplitudes of samples spaced equidistant along the time axis.

The coefficients are obtained by using computer software available from the company and programmed into the digital filter via a paper tape. The tapes are set up so that the programs can be put on a time-shared computer system. Up to 200 coefficients can be stored in the sampled-impulse-response memory.

The contents of the two memories are fed into a single multiplier section that forms the product of corresponding samples from each memory. The output of the multiplier goes to an accumulator that puts out the digitized filtered version of the input waveform.

The digital filter designed in this manner can only simulate zeroes for the filter transfer function because of its nonrecursive nature. This does not seem to limit its use though, as any filter can be represented

by its impulse response. There are few recursive filters whose impulse response cannot be satisfactorily represented by ECI's nonrecursive filter.

This concludes the discussion on SP computer implementations. FFT hardware realizations will now be discussed.

V. FFT HARDWARE

We have previously seen that a digital transfer function, $D(z)$, may be calculated by the FFT. The theory behind this was discussed, enabling the discussion of FFT hardware to now follow. Three areas of FFT hardware will be discussed starting with commercial FFT processors that are available and concluding with a discussion of the MIT fast digital processor.

Commercial Equipment

There are several manufacturers of FFT processors. One of these is the Raytheon Computer Co. of Santa Ana, California. This company manufactures what it calls an "Array Transform Processor." This processor has several capabilities, among them FFT and Inverse FFT processing, convolution integral processing, complex multiplying, complex spectral magnitude processing, real multiplying, read add/subtract processing, scanning of arrays and array movement.

The Raytheon array transform processor is an auxiliary processor of the Raytheon 700-Series computers and is a hardware array processor. It comes in several models with the models differing in the number of data points that can be handled ($256 \rightarrow 16384$) and the wordlengths available.

Another company which manufactures a hardware FFT processor is the Elsyter Co. of Syosset, New York. Their processor is labeled as the 306/HFFT. It will calculate the direct or inverse FFT and it is contained within the mainframe of its host NOVA 800 computer for more

efficiency and small size. It has a core memory of 4096-16 bit words expandable to 32768 words. It has the features of hardware multiply and divide, teletype interface, array complex coordinate converter to perform Cartesian to polar and vice-versa conversions without program intervention and an hardware FFT interface subroutine to control the operation of the hardware FFT.

It can be used in three modes: 1) Stand alone peripheral FFT processor, 2) part of a free standing spectrum analyzer system, and as 3) a free standing computer.

The last commercial FFT processor to be discussed will be the "FFT 256 FAST FOURIER TRANSFORM ANALYZER," a "stand-alone" processor manufactured by Unigon Industries, Inc., Plainview, Long Island, New York. It is a smaller processor than the one previously discussed in that it has a capacity of 256, 1024, or 4096 real points and a wordlength of 8-bits.

This processor performs the functions of the direct and inverse FFT, power spectrum and cross power spectrum analysis, square spectrum analysis, auto correlation, cross correlation, convolution, convolution spectrum and auto correlation. Each one of these functions is switch selectable. Let us now discuss a fast digital processor designed by MIT.

MIT Fast Digital Processor [27]

There are many techniques by which GP digital computers may be modified or enlarged to increase the operating speed. As an example,

speed savings may be attained by attaching fast multiply and divide hardware or by having separately addressable memory modules so that instruction cycles and data cycles may be overlapped. Increases in speed results from attaching arithmetic hardware which performs high speed special operations such as digital filtering and discrete spectrum analysis. If this arithmetic hardware is added in addition to a high speed memory, speed increases on the order of 40 to 100 can be attained in performing operation such as the FFT. This technique has a disadvantage in that it requires programming that is not easily structured, which in turn decreases the speed advantage of the special hardware.

It was because of this that emphasis was directed toward incorporating more general purpose features into a signal processing computer structure. What resulted was the MIT fast digital processor which is a general purpose digital attachment to a UNIVAC 1219 computer. The architectural changes required to increase the speed of repetitive arithmetic operations for signal processing can be classified as 1) the use of scratch pad memories, 2) pipeline schemes, and 3) parallel processing.

The fast digital processor (FDP), designed with the above architectural changes in mind, is able to perform signal processing simulations close to two orders of magnitudes faster than present conventional digital computers. As an example, a vocoder simulation which normally requires about 200 times real time on a standard computer, could be programmed to operate close to real time on the FDP.

The main applications of the FDP are in the areas of communication, radar, speech processing, biology, medicine, and sonar.

Fast commercial integrated circuit elements and a logical structure which permits each main unit of the machine to operate at maximum speed enables the FDP to obtain a speed advantage.

The arithmetic section is designed to perform efficiently the sum-of-products operations which are most important to recursive digital filtering, the FFT and correlation operations.

The data memories are structured so as to exchange data with the arithmetic section at maximum efficiency.

The control uses a separate memory for storing instructions. Its structure allows the data memories to operate at maximum speed.

Let us briefly describe some of the main features of the FDP structure.

FDP structure. Fig. 5.1 shows the most important data transfer paths of the FDP. Programs are run from the memory M^c , which controls the main data flow from memories M^a and M^b to all elements shown in Fig. 5.1. Since M^c is intended to be a small memory, longer programs can be stored in M^a and M^b and block transferred to M^c when needed. M^a , M^b , and M^c are addressed independently and can therefore be operated in parallel. Except for block transfers from M^a and M^b , the control memory M^c cannot be written into making the programs run by the FDP almost non-self-modifying.

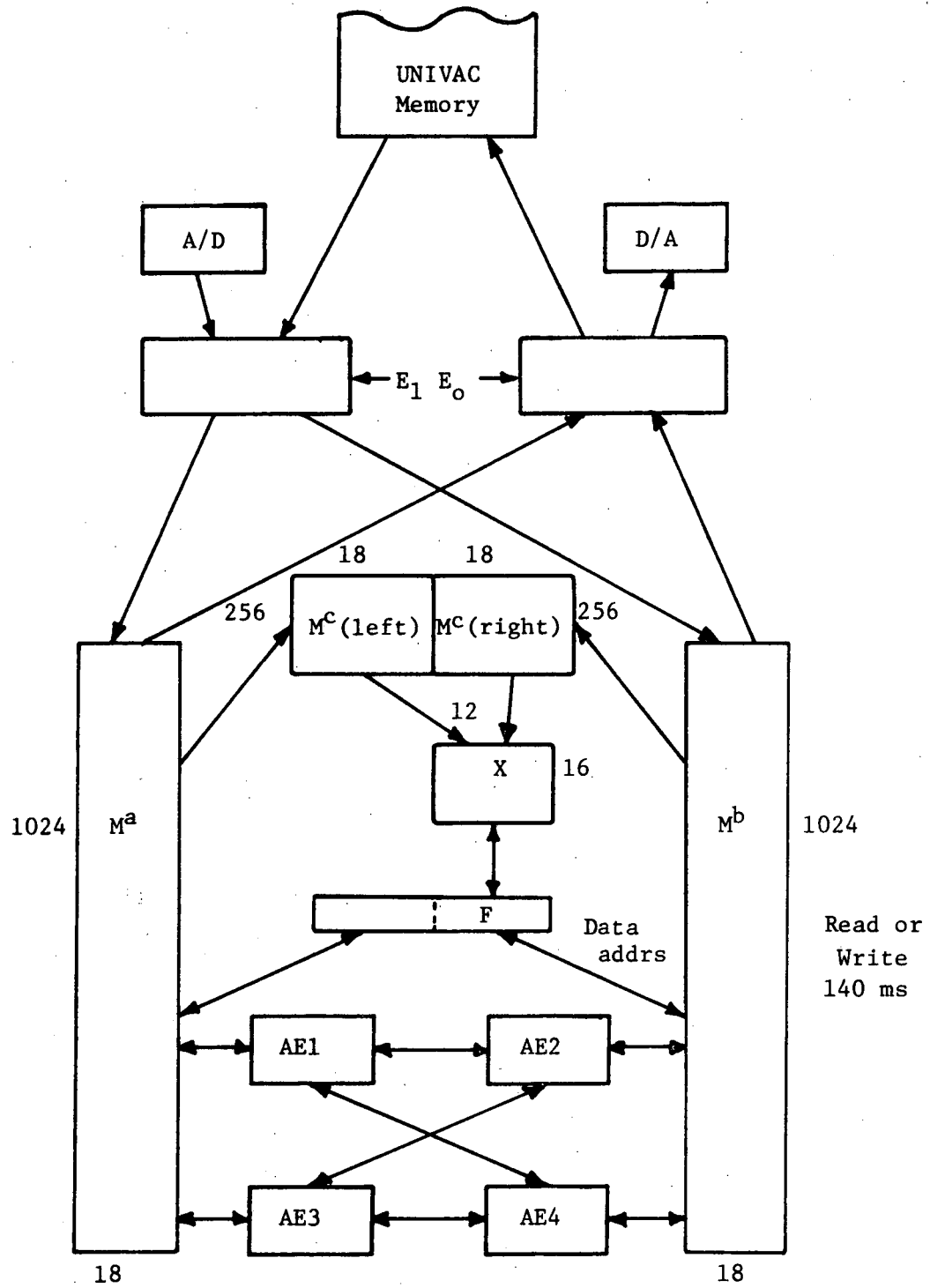


Fig. 5.1. Structure of the MIT fast digital processor.

The parallelism inherent in the FDP is partially indicated in Fig. 5.1. Listed below are a few special features incorporated for speed:

- 1) four arithmetic units, each including a multiplier which can operate in parallel with the main arithmetic registers;
- 2) two independently addressable integrated circuit memories, M^a and M^b with read and write times of 140 ns;
- 3) a separate instruction memory, which allows overlap of instructions and data cycles;
- 4) a double length instruction word which enables two instructions to be simultaneously executed on the FDP.

The size of M^a and M^b is 1024 words and is addressed by a 10-bit word. Addressing is indirect, through M^d , a 16-register 24-bit integrated-circuit memory, as is shown in Fig. 5.2. The indexed address for M^a and M^b is formed by adding the contents of the two 12-bit portions of M^d to X^a and X^b . Writing into M^d requires no special instructions because addresses 0 through 15 of M^a and M^b are wired to control M^d as well as the data memories on a write cycle.

The I/O capabilities of the FDP were minimized deliberately since the UNIVAC 1219 already supplied most of the necessary I/O control. An A/D and D/A converter were applied to make the FDP applicable for real-time processes. The only other I/O path is to and from the mother computer, the UNIVAC 1219, which will be needed for assembling and editing FDP programs and supplying medium size core storage.

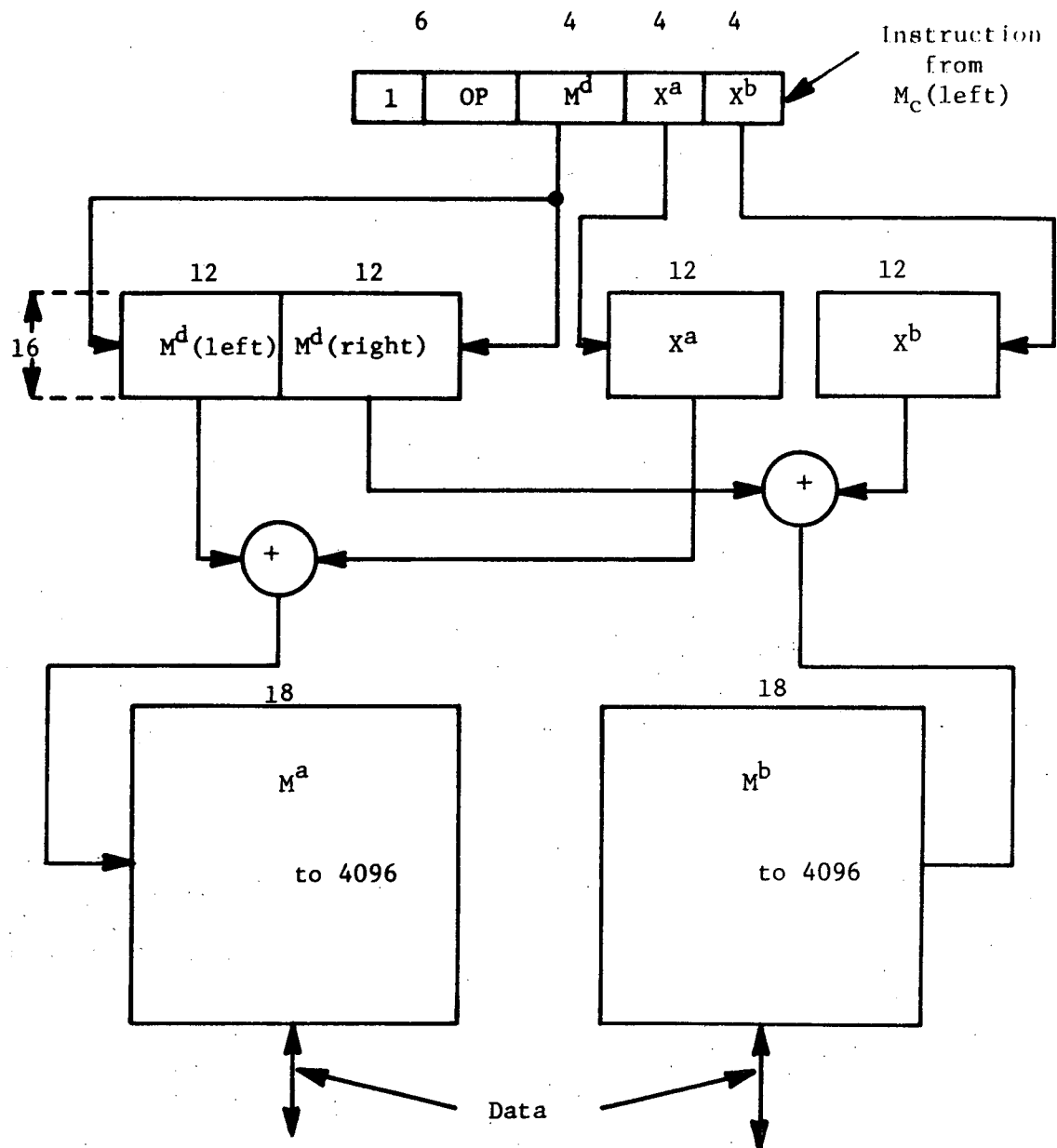


Fig. 5.2. Memory addressing.

The FDP is an 18-bit fixed point processor. Floating point routines may be used but must be programmed, as must multiple processor arithmetic.

REFERENCES

- [1] S. A. White and T. Mitsutomi, "The IC Digital Filter: A Low Cost Signal-Processing Tool," Control Engineering, pp. 58-68, June 1970.
- [2] S. C. Silver, "The Digital Filter: Potent Processing Tool," Electronic Products, pp. 32-37, March 1970.
- [3] L. B. Jackson, J. K. Kaiser, and H. S. McDonald, "Implementation of Digital Filters," IEEE International Convention - 68, New York, N. Y., March 1968.
- [4] D. J. Gawlowicz, "A Programmable Digital Compensator for Single-Loop High Performance Digital Servomechanisms," Grant NsG-36-60, Case Institute of Technology, Cleveland, Ohio, 1965.
- [5] A. Deerfield, "Canonical Digital Filters," NEREM - 67, Boston, Mass., November, 1967.
- [6] C. C. Carroll, H. T. Nagle, Jr., and H. H. Hull, "On the Realization of a Generalized Second Order Digital Compensator," Record of the IEEE 1968 Region 3 Convention, pp. 13.1.1-13.1.5, April, 1968.
- [7] H. T. Nagle, Jr., and C. C. Carroll, "Organizing a Special Purpose Computer to Realize Digital Filters for Sampled-Data Systems," IEEE Transactions on Audio and Electroacoustics (Special Issue on Digital Filters), Vol. AU-16, No. 3, September, 1968.
- [8] C. C. Carroll, J. R. Heath, and H. T. Nagle, Jr., "Reducing Quantizer Deadband with a Range Switching Digital Filter," 1969 Computer Group Conference Digest, pp. 68-81.
- [9] C. C. Carroll and J. W. Jones, Jr., "A Time-Shared Digital Filter Realization," 1969 Computer Group Conference Digest, pp. 74-77.
- [10] H. T. Nagle, Jr., C. C. Carroll, and J. W. Jones, "A Hybrid Realization for Sampled-Data Controllers," IEEE Transactions on Education, Vol. E-13, No. 1, pp. 31-37; July, 1970.
- [11] H. T. Nagle, "Digital Filter Implementations for Sampled-Data Control Systems," (INVITED PAPER) Proceedings of the 15th Midwest Symposium on Circuit Theory, Denver, Colorado; 6-7 May, 1971.
- [12] R. White and H. T. Nagle, Jr., "Digital Filter Realizations Using a Special-Purpose Stored-Program Computer," IEEE Transactions on Audio and Electroacoustics, Vol. AU-20, October, 1972, pp. 289-294.

- [13] H. T. Nagle, Jr., and C. C. Carroll, "Memory Sizing for Digital Filters," Proceedings of the IFIP Congress 71, Ljubljana, Yugoslavia; 23-25 August, 1971.
- [14] H. T. Nagle, Jr., and M. M. Edgeworth, "Computer Aided Design of Filters," TR#14, NAS8-20163, Marshall Space Flight Center, September, 1971.
- [15] F. Kuo and J. F. Kaiser, System Analysis by Digital Computer, New York, N. Y., John Wiley and Sons, Inc., 1966.
- [16] H. T. Nagle, Jr. and C. C. Carroll, "Signal Amplitude Quantization in Digital Filters," Second Hawaii International Conference on System Sciences, Honolulu, Hawaii; 22-24 January 1969.
- [17] H. T. Nagle, Jr., "An Introduction to Digital Filtering," Course Notes, Auburn University Short Course on Digital Systems and Filters, Huntsville, Alabama; 8-12 September 1969.
- [18] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An Approach to the Implementation of Digital Filters," IEEE TAU, Vol. AU-16, No. 3, September, 1968, pp. 413-421.
- [19] Series 4000 Programmable Digital Filters Specifications Sheet, Rockland Systems Corp., Blauvelt, N. Y.
- [20] Array Transform Processor Specifications Sheet, Raytheon Computer, Santa Ana, California.
- [21] Fast Fourier Transform Analyzer Specifications Sheet, Unigon Industries, Plainview, N. Y.
- [22] D. B. Kimsey and H. T. Nagle, Jr., "Digital Filter Implementation by Minicomputer," Proc. IEEE Region 3 Convention, April 10-12, 1972, pp. C3-1, C3-4.
- [23] B. C. Kuo, Analysis and Synthesis of Sampled-Data Control Systems. Englewood Cliffs, N. J.: Prentice Hall, 1963.
- [24] Y. Chu, Digital Computer Design Fundamentals, New York, N. Y., McGraw-Hill Book Company, Inc., 1962.
- [25] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, Vol. EC-13, January 1965, pp. 14-17.

- [26] L. Mattera, "Digital Filters with LSI Promise; A New World of Applications," Electronic Design, January 1971, pp. 24-26.
- [27] B. Gold, I. L. Lebow, P. G. McHugh, C. M. Rader, "The FDP, a Fast Programmable Digital Processor," IEEE Transactions on Computers, Vol. C-20, January, 1971, pp. 33-38.